



UNIVERSIDAD DEL AZUAY
FACULTAD DE CIENCIA Y TECNOLOGÍA
ESCUELA DE INGENIERÍA ELECTRÓNICA

**“Implementación de un Asistente de Lectura Audible de
Texto Impreso Mediante Visión Artificial.”**

Trabajo de graduación previo a la obtención del título de:

INGENIERO ELECTRÓNICO

Autor:

PABLO JAVIER LEÓN PARRA

Director:

Mst. GABRIEL ALFONSO DELGADO OLEAS

CUENCA, ECUADOR

2018

DEDICATORIA

Toda la dedicación y esfuerzo le dedico a Dios, por la capacidad y sabiduría que me ha brindado. También dedico este trabajo a mi familia, de manera especial a mi madre quien me ha apoyado siempre en todo sentido desde cualquier lugar, igualmente dedico a mis amigos y docentes que han contribuido para mi desarrollo profesional.

Pablo Javier León Parra

AGRADECIMIENTO

Agradezco de manera especial a mi padre por la dedicación que me ha brindado, ya que gracias a su aporte fue posible culminar con mi formación académica y personal, también agradezco a todos los docentes y amigos de todo el transcurso por la formación universitaria que han sido de vital importancia en el desarrollo académico, en especial a mi director de tesis MsC. Gabriel Delgado que siempre me ha impulsado y me ha guiado con dedicación y paciencia.

Pablo Javier León Parra

INDICE DE CONTENIDOS

| | |
|---|-------------------------------|
| DEDICATORIA | ii |
| AGRADECIMIENTO | iii |
| ÍNDICE DE FIGURAS | viii |
| ÍNDICE DE TABLAS | x |
| ÍNDICE DE ANEXOS | xi |
| RESUMEN | ¡Error! Marcador no definido. |
| ABSTRACT | ¡Error! Marcador no definido. |
| INTRODUCCIÓN | 1 |
| CAPÍTULO 1: INVESTIGACIÓN DE TÉCNICAS, SOFTWARE Y DISPOSITIVOS DE LECTURA. | 2 |
| 1.1. Introducción..... | 2 |
| 1.2. Estado del arte | 2 |
| 1.2.1. Lectura tecnológica auditiva | 3 |
| 1.2.2. Maquina delta..... | 3 |
| 1.2.3. Revisores de pantalla..... | 3 |
| 1.2.4. Dibujos en relieve | 3 |
| 1.2.5. Línea braille | 4 |
| 1.2.6. Anotadores electrónicos | 4 |
| 1.2.7. Orcam..... | 5 |
| 1.3. Marco teórico | 6 |
| 1.3.1. Lectura..... | 6 |
| 1.3.1.1. Lectura visual | 6 |
| 1.3.1.2. Lectura braille | 6 |
| 1.3.2. Visión artificial..... | 7 |
| 1.3.3. Digitalización de textos..... | 8 |

| | | |
|--|--|-----------|
| 1.3.4. | Reconocimiento óptico de caracteres..... | 8 |
| 1.3.4.1. | Historia..... | 9 |
| 1.3.4.2. | Funcionamiento..... | 10 |
| 1.3.4.3. | Aplicaciones..... | 14 |
| 1.3.5. | Text-To-Speech (TTS)..... | 15 |
| 1.3.5.1. | Funcionamiento..... | 16 |
| 1.3.5.2. | Aplicaciones..... | 18 |
| 1.3.6. | Sistema embebido..... | 18 |
| 1.3.6.1. | Características..... | 18 |
| 1.3.6.2. | Aplicaciones..... | 19 |
| 1.3.6.3. | Ejemplos de sistemas embebidos..... | 20 |
| 1.3.6.4. | Programación en sistemas embebidos..... | 21 |
| 1.3.7. | Lenguaje de programación Python..... | 22 |
| 1.3.8. | OpenCV..... | 23 |
| 1.4. | Conclusiones..... | 24 |
| CAPÍTULO 2: DISEÑO DEL SOFTWARE ENCARGADO DE ADQUIRIR, SELECCIONAR Y ALMACENAR LAS IMÁGENES. | | 26 |
| 2.1. | Introducción..... | 26 |
| 2.2. | Módulos utilizados..... | 26 |
| 2.2.1. | PIL..... | 26 |
| 2.2.2. | NumPy..... | 26 |
| 2.2.3. | Os..... | 26 |
| 2.2.4. | Imutils..... | 27 |
| 2.3. | Marco teórico..... | 27 |
| 2.3.1. | Imagen digital..... | 27 |
| 2.3.2. | Resolución espacial..... | 29 |
| 2.3.3. | Resolución de niveles..... | 29 |

| | | |
|---|--|-----------|
| 2.4. | Desarrollo de software..... | 29 |
| 2.4.1. | Captura de la imagen..... | 29 |
| 2.4.2. | Segmentación de la imagen..... | 30 |
| 2.4.3. | Selección de la imagen..... | 34 |
| 2.4.4. | Almacenamiento de las imágenes | 35 |
| 2.5. | Conclusiones | 36 |
| CAPÍTULO 3: INTERPRETACIÓN DEL TEXTO Y TRADUCCIÓN A VOZ SINÉTICA. | | 37 |
| 3.1. | Introducción..... | 37 |
| 3.2. | Módulos utilizados | 37 |
| 3.2.1. | Pytesseract..... | 37 |
| 3.2.2. | Os.system | 40 |
| 3.2.3. | Os.system (lame)..... | 40 |
| 3.2.4. | Espeak | 40 |
| 3.2.5. | Gtts | 41 |
| 3.2.6. | Pygame | 42 |
| 3.2.7. | Time | 42 |
| 3.3. | Marco teórico | 42 |
| 3.3.1. | Filtrado de preprocesado | 42 |
| 3.3.1.1. | Transformación de nivel de gris..... | 42 |
| 3.3.1.2. | Filtrado Binarización de Otsu..... | 42 |
| 3.3.2. | Codificación | 43 |
| 3.4. | Desarrollo del software | 45 |
| 3.4.1. | Filtrado de la imagen..... | 45 |
| 3.4.2. | Obtención del texto de la imagen..... | 50 |
| 3.4.3. | Generación de la voz a través del sistema TTS..... | 53 |

| | | |
|---|-----------------------------------|-----------|
| 3.5. | Conclusiones | 55 |
| CAPÍTULO 4: COMPROBACIÓN E INTEGRACIÓN DE LOS SISTEMAS IMPLEMENTADOS. | | 57 |
| 4.1. | Introducción..... | 57 |
| 4.2. | Módulos utilizados | 57 |
| 4.2.1. | GPIO | 57 |
| 4.3. | Marco teórico | 57 |
| 4.3.1. | Fuentes de textos | 57 |
| 4.4. | Desarrollo del software | 58 |
| 4.4.1. | Integración del sistema..... | 58 |
| 4.4.2. | Desarrollo del Hardware | 63 |
| 4.4.3. | Funcionamiento del sistema..... | 66 |
| 4.5. | Conclusiones y limitaciones | 69 |
| CONCLUSIONES..... | | 72 |
| RECOMENDACIONES..... | | 73 |
| REFERENCIAS | | 75 |
| ANEXOS | | 83 |
| Anexo 1: Diagrama de flujo | | 83 |

ÍNDICE DE FIGURAS

| | |
|---|----|
| Figura 1.1: Estructura de una red neuronal multicapa feedforward | 10 |
| Figura 1.2: Modelo de una neurona | 11 |
| Figura 1.3: Ejemplo de una muestra del patrón "1" y el resultado de su identificación. | 13 |
| Figura 1.4: Proceso de conversión de texto a voz..... | 16 |
| Figura 1.5: Raspberry Pi 3, modelo B+ | 21 |
| Figura 2.1: Representación de un pixel en una imagen. | 27 |
| Figura 2.2: Modos de color de las imágenes..... | 28 |
| Figura 2.3: Idle de Python 3.4, algoritmo para realizar la captura de la imagen de forma manual..... | 30 |
| Figura 2.4: Idle de Python 3.4, algoritmo para realizar la detección de contornos.... | 31 |
| Figura 2.5: Idle de Python 3.4, resultado de la segmentación de la imagen y calibración del perímetro de la imagen..... | 32 |
| Figura 2.6: Idle de Python 3.4, riesgo de error en la segmentación..... | 33 |
| Figura 2.7: Idle Python 3.4, corrección de error en la segmentación de la imagen. .. | 34 |
| Figura 2.8: Idle Python 3.4, selección de la imagen. | 34 |
| Figura 2.9: Idle Python 3.4, algoritmo para almacenamiento de las imágenes..... | 35 |
| Figura 3.1: Comparación en motores OCR..... | 38 |
| Figura 3.2: Ancho variable de caracteres (proporcional) y ancho fijo..... | 39 |
| Figura 3.3: Binarización de Otsu. | 43 |
| Figura 3.4: Codificación de caracteres..... | 44 |
| Figura 3.5: Codificación UTF-8 y Latin-1..... | 45 |
| Figura 3.6: Idle de Python 3.4, resultado del Filtrado Gausseano. | 46 |
| Figura 3.7: Idle de Python 3.4, resultado de operaciones morfológicas. | 47 |
| Figura 3.8: Idle de Python 3.4, preprocesado de la imagen. | 48 |
| Figura 3.9: Idle de Python 3.4, error en el filtrado a causa de captura errónea. | 49 |
| Figura 3.10: Idle de Python 3.4, limitaciones de tipo de textos. | 49 |
| Figura 3.11: Idle de Python 3.4, obtención de texto de una hoja tamaño A4. | 51 |
| Figura 3.12: Idle de Python, textos con fondos a color..... | 52 |

| | |
|---|----|
| Figura 3.13: Idle de Python 3.4, texto en hoja de tamaño A5. | 52 |
| Figura 3.14: Idle de Python 3.4, algoritmo de generación del voz. | 53 |
| Figura 3.15: Idle de Python 3.4, algoritmo de reproducción de audios. | 54 |
| Figura 4.1: Idle de Python 3.4, algoritmo de acercamiento al texto. | 58 |
| Figura 4.2: Idle de Python, algoritmo de captura automática. | 59 |
| Figura 4.3: Idle de Python, algoritmo de ayudas audibles al seguimiento de la imagen. | 59 |
| Figura 4.4: Idle de Python, algoritmo de integración de los sistemas..... | 60 |
| Figura 4.5: Idle de Python 3.4, pruebas sobre textos de diferentes anchos..... | 61 |
| Figura 4.6: Idle de Python 3.4, letras delgadas pierden sus atributos. | 62 |
| Figura 4.7: Idle de Python, resultados finales con un error despreciable. | 62 |
| Figura 4.8: Software Multisim, simulación del circuito de control. | 63 |
| Figura 4.9: Software Ultiboard, diseño de PCB..... | 64 |
| Figura 4.10: Software Ultiboard, diseño impreso del PCB..... | 64 |
| Figura 4.11: Idle de Python 3.4, asignación de pines GPIO. | 65 |
| Figura 4.12: Conexión de la placa del control al sistema embebido..... | 66 |
| Figura 4.13: Idle de Python 3.4, implementación del menú. | 66 |
| Figura 4.14: Idle de Python 3.4, implementación de las funciones correspondientes al menú..... | 67 |
| Figura 4.15: Idle de Python 3.4, algoritmo para pulsantes de control. | 68 |

ÍNDICE DE TABLAS

| | |
|---|----|
| Tabla 1.1. Comparación de software que utilizan OCR. | 14 |
| Tabla 1.2 Lenguajes de programación en sistemas embebidos. | 22 |

ÍNDICE DE ANEXOS

Anexo 1: Diagrama de flujo83

"Implementación de un Asistente de Lectura Audible de Texto Impreso Mediante Visión Artificial."

RESUMEN

En el presente documento se realiza la implementación de un dispositivo capaz de leer un texto específico impreso en papel mediante visión artificial e interpretarlo mediante una voz sintética. El cual, inicialmente toma una imagen de la hoja que contiene el texto a través de una cámara. Luego se procesa la imagen para su posterior etapa en la cual mediante el sistema "Optical Character Recognition" (OCR) se obtiene el texto. Finalmente, con la ayuda de un software sintetizador de voz o también conocido como "Text To Speech" (TTS) se representa el texto en audio. De esta manera, se facilita el acceso a la información escrita y mejora la vida de muchas personas que tienen dificultad en la lectura.

Palabras clave: Visión artificial, OCR, TTS.



Director de Escuela

Ing. Daniel Iturralde Piedra PhD



Director de Trabajo de graduación

MSc. Gabriel Alfonso Delgado Oleas



Autor de Trabajo de graduación

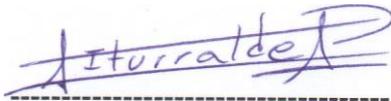
Pablo Javier León Parra

"Implementation of an Audible Reading Assistant for Printed Text Through Artificial Vision."

ABSTRACT

This investigation aimed to implement a device capable of reading a specific text printed on paper by using artificial vision. The paper was read using a synthetic voice. Initially, an image of the sheet containing the text is taken through a camera. Then, this image is processed for the next stage, where the text is obtained through the "Optical Character Recognition" (OCR) system. Finally, the audio text is represented by using a speech synthesizer software known as "Text To Speech" (TTS). This facilitated access to written information and improved the lives of many people with reading difficulties.

Keywords: Artificial Vision, OCR, TTS.



Faculty Director

Ing. Daniel Iturralde Piedra PhD



Thesis Director

MsC. Gabriel Alfonso Delgado Oleas



Author

Pablo Javier Leon Parra



Translated by
Ing. Paul Arpi

INTRODUCCIÓN

¿Has pensado cómo hubiera sido tu vida sin la lectura? La lectura es importante para el desarrollo de la educación personal, pero, existen personas que no pueden leer por algún tipo de enfermedad o por carencia de visión. Además, hasta agosto del año 2018, el 11,88% de las personas registradas en el Consejo Nacional para la Igualdad de Discapacidades (CONADIS), tienen discapacidad visual, de las cuales más de 2600 personas viven en el cantón Cuenca (CONADIS, 2018, agosto). Por lo tanto, estas personas se ven limitadas a la información impresa (documentos), por ello se les dificulta desenvolverse con dependencia dentro del mundo cotidiano.

Gracias al avance y crecimiento tecnológico que se vive en la actualidad, hoy en día se cuenta con una gran variedad de dispositivos y herramientas tecnológicas que les permiten a las personas facilitar sus labores o solucionar ciertos inconvenientes que se presentan en la vida. Del mismo modo se necesita un sistema que aporte a todas las personas que tengan dificultad para leer.

Por lo tanto, se realizará un breve estudio y recopilación de la tecnología existente para seleccionar las herramientas adecuadas para trabajar, luego se realizará un programa que capture las imágenes y las procese para su posterior interpretación en audio. Finalmente se realizará la integración de los sistemas y pruebas de campo para poder realizar un ayudante de lectura mediante visión artificial y romper las barreras que limitan a todas estas personas acceder a la información escrita sin la ayuda de otra persona, es decir que puedan saber lo que dice un texto impreso de manera autónoma. En definitiva, facilitaría el desempeño de su vida con el entorno, además de contribuir a su desarrollo personal, formativo y laboral, cambiando por completo la vida del usuario.

CAPÍTULO 1: INVESTIGACIÓN DE TÉCNICAS, SOFTWARE Y DISPOSITIVOS DE LECTURA.

1.1. Introducción

La lectura es considerada como una actividad dentro de las más importantes y con mayor utilidad que puede desempeñar el ser humano en su vida. Además, es una acción que define al ser humano, puesto que, son los únicos seres vivos que desarrollan un sistema intelectual y generalmente desde temprana edad. De igual manera, la educación de una persona puede comenzar a crecer gracias a la lectura y a su vez requiere concentración, atención y compromiso para mejorar sus resultados, permitiendo reflexionar sobre ideas, alentar la imaginación, mejorar la ortografía, etc. A su vez, la lectura contribuye a una formación académica, de tal manera que la persona pueda desempeñar empleos sencillos o complejos en un futuro (Bembibre, 2011).

Por otra parte, se conoce que la tecnología es fundamental para el desarrollo en la sociedad, puesto que brinda la comunicación entre personas, producen una mayor cantidad de alimentos, transportan personas con mayor facilidad, producen productos de gran utilidad, etc. También gracias a los dispositivos tecnológicos se puede realizar tareas laboriosas, complejas y repetitivas con mayor facilidad, haciendo de esta manera la vida más cómoda y segura (Martinez, 2016).

Por lo tanto, se realiza un estudio de las diferentes técnicas de lectura o medios de acceso a la información y dispositivos de lectura existentes, con el fin de tener una visión más amplia o con mayor facilidad para adaptar la tecnología existente en construir un dispositivo capaz de ayudar a la lectura, facilitando el acceso a textos específicos impresos en papel.

1.2. Estado del arte

A lo largo de la historia se han desarrollado algunos dispositivos para acceder a la información escrita, es por ello por lo que a continuación vale la pena destacar algunos de estos dispositivos con el fin de tener una mayor idea de la tecnología existente.

1.2.1. Lectura tecnológica auditiva

Existe un recurso muy óptimo para la lectura de textos como es la lectura tecnológica auditiva, en la cual la persona escucha el texto que anteriormente o en ese momento se presenta escrito. En el caso que sea un texto presentado con anterioridad se denomina libro hablado o parlante, que previamente se traduce a lenguaje oral un libro escrito en letra tradicional, por lo tanto, dichas grabaciones facilitan el cansancio que supone la lectura táctil. En 1988 se indicó este método como idóneo ya que genera mayor velocidad con un rango de hasta 210 palabras por minuto, por ello es un método muy utilizado en la actualidad, por lo que instituciones competentes como la Organización Nacional de Ciegos Españoles (ONCE), se esfuerzan por traducir la mayor cantidad de obras, ya que existe una limitación de otras traducidas (Rodríguez Fuentes, 2005).

1.2.2. Máquina delta

La máquina Delta, consta de un dispositivo portátil que lee un documento impreso a través de una cámara sobrepuesta al texto y lo representa en una línea de 12 caracteres en sistema braille, que va actualizándose con la velocidad del movimiento de la cámara. Los textos deben tener la característica que no pueden ser manuscritos ni estar reflejados en malas impresiones, por el contrario, se necesita que sean textos de calidad con letras entre 1 a 6 milímetros y se necesita en algunos casos la asistencia de una persona vidente para el entrenamiento de la máquina para que reconozca los caracteres (Martos Navarro & Muñoz Labiano, 2008).

1.2.3. Revisores de pantalla

Los sintetizadores de voz que permiten reproducir lo que dice en la pantalla de la computadora en forma de secuencia de caracteres, informando a la persona no vidente el material reflejado en el archivo. Por ejemplo, PcCoz, Orpheus, Apollo2, etc. Igualmente, los lectores de pantalla verbalizan lo que encuentran en la pantalla, incluyendo las descripciones de los botones, menús, signos de puntuación, etc. Algunos programas que realizan estas funciones pueden ser Hal, Windows-Eyes, Jaws, etc. (Serrano Mascaraque, 2008).

1.2.4. Dibujos en relieve

Existe una máquina conocida como *Termoform* que copia textos escritos en papel a textos en relieve utilizando plásticos sometidos a altas temperaturas (Rodríguez & Gallego Granda, 2001). Para este proceso se utiliza sobre una matriz en relieve una hoja plástica que posee características especiales. La matriz puede ser de diferentes materiales, tales como: maderas, cuerdas, lijas, metales, cartones, etc. Por lo tanto, con el *Termoform* se puede sacar copias en relieve plasmado en el plástico, colocando el plástico sobre la matriz con ayuda de una bomba al vacío y sometiéndola a temperaturas elevadas. Luego de un tiempo se retira el calor y ya se puede retirar el plástico de la matriz (Carrio Díaz, 2006).

También existe un papel llamado micro cápsula, muy utilizado por su sencillez, que se utiliza para realizar diseños ya sea a mano alzada o diseños en un ordenador, pero por lo general en color negro para luego someterlas a calor con el objetivo de que absorban el calor y que se dilaten. Luego de esto, con el relieve, las personas no videntes pueden explorar el diseño. Para diferenciar áreas, objetos, límites se utiliza diferentes formas de líneas, diferentes grosores, distintos rellenos de zonas como puntos o cuadrados (Carrio Díaz, 2006).

1.2.5. Línea braille

El *display braille* es un equipo de sobremesa que genera una salida táctil del texto presentado en la pantalla de un computador, en líneas de hasta ochenta ocurrencias (Serrano Mascaraque, 2008). La Línea braille presenta una desventaja considerable, que puede representar como máximo, una línea de pantalla. Este método puede ser la vía exclusiva para las personas no videntes que tienen discapacidad auditiva también. Pero es muy aceptado en los niveles elementales de enseñanza (Carrio Díaz, 2006).

Por otra parte, para las personas que tienen dificultades de visión también son utilizadas las impresoras braille, las cuales imprimen en puntos la información que envía el ordenador. Las impresoras constan de barras de punzones que deforman el papel (Carrio Díaz, 2006).

1.2.6. Anotadores electrónicos

Consiste en un equipo portátil autónomo con capacidades de manejo sencillo de un promedio de 2 Mb. El control del dispositivo y la lectura son mediante síntesis de voz y mediante su teclado braille se pueden crear archivos de texto, acceder a la calculadora científica, agenda, etc. Además, gracias a su procesador de textos, permite elaborar documentos que se pueden guardar, imprimir en braille o en impresoras ordinarias o a su vez enviarlos a un ordenador o un dispositivo de almacenamiento. Los anotadores también pueden ser con teclado QWERTY, u otros que son de mayor peso y volumen, que se asemejan a un ordenador personal (Carrio Díaz, 2006).

1.2.7. Orcam

Una empresa israelí fundada en 2010 con el objetivo contribuir en la ayuda de los discapacitados desarrolla un sistema llamado Orcam, de reconocimiento de texto, números de autobús, lugares, artículos de prensa, etc. También reconoce objetos que se indiquen en una cámara montada sobre unas gafas, simplificando la utilización de smartphones. Por lo que se obtiene información del entorno con solo marcar con el dedo el texto o el párrafo que se desea conocer (Sánchez, 2013). Orcam aprovecha la visión artificial en una plataforma portátil para optimizar la vida de personas con discapacidad visual, fue fundada por el CTO Profesor Amnon Shashua y el CEO Ziv Aviram, y fue lanzado en Estados Unidos en el 2015 (OrCam, 2018).

Este dispositivo proporciona información mediante audio en un altavoz, de lo captado por una cámara diminuta que se puede conectar a diferentes tipos de gafas. Este dispositivo utiliza inteligencia artificial para reconocer rostros, leer textos, e identificar productos (OrCam, 2018).

Existe hasta el 2018 principalmente 2 tipos de productos; el primero se trata de “OrCam MyReader”, ayuda a personas con dificultades para leer ya que se enfoca básicamente en la lectura de textos, con un costo aproximado de 3500 euros; el segundo producto, “OrCam MyEye”, también puede leer textos y además puede realizar reconocimiento facial, informar la hora y fecha, identificar billetes, colores, lectura en idiomas, identificación de código de barras, etc., a un costo aproximado de 4500 euros (OrCam, 2018).

1.3. Marco teórico

Es importante conocer el sustento teórico para ampliar las herramientas con las que se van a sustentar el dispositivo a construir, por lo tanto, hace un estudio de los conceptos involucrados con los medios que se utilizan para acceder a la información. También se abordará el software indicado para implementar el asistente de lectura y el dispositivo idóneo para soportar el programa.

1.3.1. Lectura

Se conoce como lectura a la acción de comprender una representación gráfica de los caracteres empleados, ya sea pasando la vista por un escrito o de un modo determinado (Diccionario de la lengua española, 2017). También se conoce la lectura como la apreciación de la información mediante ciertos códigos, ya sean visuales, auditivos o táctiles (Pérez Porto & Gardey, 2012).

1.3.1.1. Lectura visual

La lectura visual es un método muy utilizado que empieza a estudiarse desde 1900, con el análisis de la percepción de la decodificación de las formas simples. Además, se identifica a la lectura visual como un proceso multimodal con marcos de lectura establecidos, desde la parte izquierda del texto hacia la derecha y desde la parte superior hasta la inferior (Zaganelli, 2014). En consecuencia, existe un parámetro muy considerado en la lectura, que se conoce como la velocidad lectora, la cual representa la cantidad de palabras que se consiguen leer por minuto, teniendo en consideración que se pretende comprender el contenido del texto que se lee. La velocidad adecuada está en el rango de 200 a 300 palabras por minuto, pero el leer más rápido ayuda a comprender de mejor manera el texto, puesto que, el cerebro mientras espera palabras se mantiene en el tema y no infieren otros pensamientos que desvíen la atención y hagan olvidar lo que se está leyendo (Biblioteca de la Universidad de Extremadura, 2018).

1.3.1.2. Lectura braille

El sistema braille en sus líneas generales nace en el año 1925 por el francés Louis Braille, quien sufrió un accidente en los primeros años de su niñez. La escritura Braille

en castellano pretende reflejar la escritura en caracteres latinos ya que se basa en un código táctil de letras, signos de puntuación, signos de entonación, etc. En sí consiste en un alfabeto reconocido de forma internacional. El sistema braille es muy utilizado ya que facilita el acceso a la comunicación, recreo, estudio y creación intelectual especialmente a personas con dificultades de visión, pero también tiene ciertas desventajas, ya que, puede ser dificultoso, lento, fatigante y poco práctico (Fernández del Campo, 2001). Además, el braille al necesitar un aprendizaje previo no es accesible para todas las personas no videntes, por lo que se calcula que en el año 1994 solo el 10% de la población de personas no videntes utilizaban este lenguaje (Burger, 1994). En el año 2012 se aproxima que menos del 1% de la población no vidente en Reino Unido utilizan braille (Rose, 2012). Otro motivo que dificulta el acceso braille, es el punto de vista práctico, ya que como ejemplo se estima que un diccionario como “El Pequeño Larousse” en braille requiere 15 metros lineales en estantería, que se requiere un aproximado de 157 volúmenes (Burger, 1994).

1.3.2. Visión artificial

La visión artificial puede extraer mucha información del mundo físico como brillo, colores, formas, texturas, etcétera; a través de imágenes por medio de sensores como cámaras, que posteriormente por lo general son procesadas en un computador (Vélez Serrano, Moreno Díaz, Sánchez Calle, & Sánchez Marín, 2003). Ya que luego de la adquisición de la imagen los datos obtenidos pasan a procesarse, por lo que se dividen en varias partes para extraer las características necesarias y después de un reconocimiento se interpretan y localizan en un medio 'digitalizado' (Alvear Puertas, Rosero Montalvo, Peluffo Ordóñez, & Pijal Rojas, 2017). De igual manera que el ser humano captura la luz a través de los ojos, la misma que llega hasta el cerebro donde se procesa y actúa después de descomponer la imagen; también la visión artificial intenta reproducir este comportamiento que se reflejan en cuatro fases principales:

- 1) Es una etapa puramente sensorial en donde se adquiere las imágenes mediante algún tipo de sensor.

- 2) Consiste en el tratamiento digital de las imágenes para facilitar las etapas posteriores en donde se eliminan partes indeseables de las imágenes o se destacan partes interesadas mediante procesos como filtros o transformaciones geométricas.
- 3) La tercera etapa se conoce como segmentación, que separa los elementos destacados de una imagen para comprenderla.
- 4) La etapa final se denomina etapa de reconocimiento o clasificación, en donde se distinguen los objetos segmentados tomando como referencia ciertas características establecidas previamente para diferenciarlos.

El orden de estas etapas en ocasiones puede variar o se realiza hacia atrás. También se puede volver a alguna etapa, como regresar a la etapa de segmentación si falla la etapa de reconocimiento o incluso a la de captura si falla alguna de las etapas siguientes (Vélez Serrano, Moreno Díaz, Sánchez Calle, & Sánchez Marín, 2003).

1.3.3. Digitalización de textos

Trata de un proceso en el cual podemos transformar un documento impreso a un documento en formato electrónico. Este proceso ofrece grandes beneficios como, por ejemplo: mejora la comunicación interna, conversión de formatos, permite que un mayor número de personas accedan a esa información en un mismo tiempo, etc. Se utilizan diferentes medios para lograrlo, sin embargo, un método muy utilizado es capturar mediante un escáner el texto deseado en forma de imagen y posteriormente a través un software obtener los textos que contienen las imágenes. Hay que tener en cuenta elementos importantes para este proceso, como lo son: el tipo de resolución del dispositivo que captura la imagen, manejo del contraste, reconocimiento óptico de caracteres (OCR), capacidad para comprimir formatos, corrección de errores, etc. (Pérez González, Lara Lemus, & Naranjo Hernández, 2004).

1.3.4. Reconocimiento óptico de caracteres

El reconocimiento óptico de caracteres o llamado por sus siglas en inglés “Optical Character Recognition (OCR)”, atrae mucho interés por investigadores pese a que los algoritmos son complicados y se requiere un hardware potente para su implementación (Quoc Vuong & Ngoc Do, 2014). El OCR, simula la habilidad humana para el reconocimiento de patrones alfanuméricos, esto lo logra creando o utilizando modelos físicos y matemáticos (Gutierrez, Frydson, & Phd. Vintimilla, 2011).

1.3.4.1. Historia

En 1929 en Alemania se entregó la primera patente sobre OCR a Gustav Tauschek y en 1935 en EE. UU. La máquina de Gustav Tauschek consistía en un dispositivo mecánico en base de plantillas y utilizaba un foto-detector para dirigir la luz hacia ellos cuando el carácter se reconociera y se encuentren alineados. En el año de 1955 Readers Digest instaló el primer sistema comercial, que años después fue donado al museo Smithsonian. En 1965 en Reino Unido, se utilizó la tecnología OCR para un sistema completo de actividades bancarias, el cual revolucionó el sistema de pago de cuentas. En Canadá utilizan desde 1971 el sistema de OCR para leer los nombres y las direcciones en el correo postal y con esa información imprimen un código de barras basado en el código postal (Ordóñez L., 2009)

En la actualidad los sistemas OCR se consideran con un 99% de exactitud, aunque hay ocasiones que se requiere la revisión humana para los errores. Por otra parte, cuando se trata de caracteres impresos a mano, el índice de exactitud está entre el 80 al 90%, por lo cual es un poco limitado este recurso. Además, las medidas de reconocimiento del texto cursivo son más bajas que las del reconociendo de texto impreso a mano. También cabe recalcar, que es más sencillo el reconocimiento de palabras enteras que el de caracteres individuales, ya que es de ayuda la información del contexto. Así también, el tener información de la gramática de la lengua involucrada en el texto, es de mucha ayuda, ya que se puede determinar si la palabra a reconocer es verbo o sustantivo (Ordóñez L., 2009).

1.3.4.2. Funcionamiento

Entre los modelos físicos y matemáticos que utiliza el OCR para el reconocimiento de patrones alfanuméricos, se destacan los más conocidos (Gutierrez, Frydson, & Phd. Vintimilla, 2011):

- 1) Redes neuronales: se basa en el funcionamiento del cerebro humano, la solución se la da mediante la evolución de un sistema “inteligente”, más no como pasos predeterminados (Gutierrez, Frydson, & Phd. Vintimilla, 2011). También tiene cierta semejanza al cerebro, ya que, se adquiere el conocimiento mediante la red basado en un proceso de aprendizaje y las fuerzas o pesos sinápticos con las que se interconectan las neuronas son utilizados para guardar información (Aldabas-Rubira, 2002). Son múltiples las variedades de aplicaciones y de tipos de redes neuronales, sin embargo, entre las de reconocimiento de formas, está la red multicapa. En el caso de tener conexiones unidireccionales y hacia adelante se le conoce como “feedforward”, esta estructura se indica en la siguiente figura (Ordóñez L., 2009).

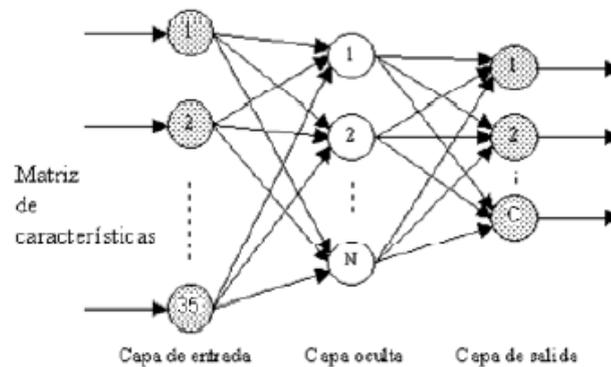


Figura 1.1. Estructura de una red neuronal multicapa feedforward

Fuente: (Ordóñez L., 2009)

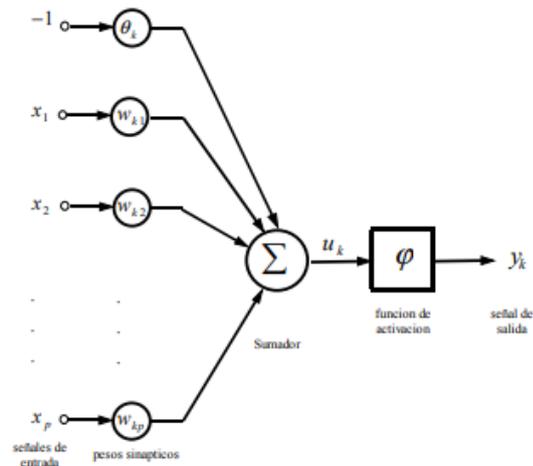


Figura 1.2. Modelo de una neurona

Fuente: (Aldabas-Rubira, 2002)

La red feedforward consta de la capa de entrada compuesta por un conjunto de nodos de entrada, un conjunto de capas ocultas de neuronas que pueden ser una o más y la capa de salida. El funcionamiento se le llama “Multi Layer Perceptrons (MLP)”, que consiste en que la señal de entrada, entra por la capa de entrada, pasa por la capa oculta y va hasta la capa de salida (Aldabas-Rubira, 2002).

La resolución de problemas se basa en el aprendizaje que se le da a la red neuronal, se puede basar en la regla “back propagation”, que consiste en realizar dos pasadas por las capas de la red, una vez hacia adelante aplicando un patrón o vector de entrada, que se propaga a través de las capas hasta obtener el vector de salida con los pesos sinápticos fijos y la segunda pasada hacia atrás, en donde los pesos si se modifican siguiendo a la regla de corrección de error. En consecuencia, la señal de error es la comparación de la señal de salida con la señal deseada. La señal error se propaga en dirección contraria por toda la red para modificar los pesos de tal manera que cuando se vuelva a pasar el vector de entrada, se obtenga una señal muy aproximada a la señal deseada (Aldabas-Rubira, 2002).

Por lo tanto, las características de un perceptrón multicapa son:

- Como se puede apreciar en la ilustración 2, el modelo de una neurona incluye una función no lineal. Donde la salida “y” de una neurona “k”, es la función sigmoide, siendo una función continua y derivable que está relacionado por:

$$y_k = \frac{1}{1+\exp(-u_k)} \quad (1)$$

Siendo “ u_k ” el resultado total de la suma de la actividad interna en la neurona (Aldabas-Rubira, 2002).

- La red está conformada por una o más capas ocultas intermedias, las cuales capacitan progresivamente a la red almacenando internamente la información (Aldabas-Rubira, 2002).
- Existe un gran número de conexiones determinadas por los pesos de la red (Aldabas-Rubira, 2002).

A continuación, se muestra un ejemplo de reconocimiento óptico de caracteres utilizando el entrenamiento “back propagation” a un perceptrón multinivel. Entonces, se pretende clasificar los números del 0 al 9, el punto y el guion. Se tiene un panel de entrada por una matriz de 7x5. En consecuencia, se obtienen 35 entradas correspondientes a los puntos de la matriz, en donde, si el valor es 0 corresponde al punto blanco y si el punto es negro obtendrá un valor de 1. Como se puede observar en la siguiente ilustración, se toma el patrón “1”, y la visualización de la numeración de la matriz. Las salidas están representadas por las posibilidades de coincidencias, en este caso se presentan 12 salidas posibles de 12 patrones posibles, activándose únicamente la salida a la clase que pertenezca con su salida próximo a uno, ya que las otras tendrán valores próximos a cero y se mantendrán desactivadas (Aldabas-Rubira, 2002).

1.3.4.3. Aplicaciones

Existen muchas aplicaciones que se basan en el uso de la tecnología OCR, entre las cuales poseen diferentes tipos de licencias de uso, como, por ejemplo: uso comercial, que es desarrollado con el fin de lucrar con la utilización del software; software libre, su utilización, modificación y distribución es libre, en algunos casos un software puede ser vendido; código abierto, pone a disposición su código fuente para cualquier usuario y de manera gratuita; freeware, estos programas permiten la redistribución pero no la modificación, no son software libre pero es gratuito (Tecnología informática, s.f.). Los softwares también pueden ser diseñados con determinadas compatibilidades, por ello, a continuación, se indica una tabla comparativa entre las aplicaciones más utilizadas, su tipo de licencia de uso y compatibilidad:

Tabla 1.1. Comparación de software que utilizan OCR.

| Software | Licencia | Compatibilidad |
|---|-----------------|---|
| ExperVision TypeReader & OpenRTK | Comercial | Windows, Mac OS X, Unix, Linux, OS/2 |
| ABBYY Fine Reader OCR | Comercial | Windows |
| OmniPage | Comercial | Windows, Mac OS |
| Readiris | Comercial | Windows, Mac OS |
| SmartZone (formerly known as Zonal OCR) | Comercial | Windows |
| Computhink's ViewWise | Comercial | Windows |

| | | |
|-----------------------------------|----------------------|--------------------------------|
| Microsoft Office Document Imaging | Comercial | Windows, Mac OS X |
| Microsoft Office OneNote 2007 | Comercial | Windows |
| Ocrad | Software libre | Unix-like, OS/2 |
| Brainware | Comercial | Windows |
| HOCR | Software libre | Linux |
| OCROPUS | Código abierto | Linux |
| ReadSoft | Comercial | Windows |
| SimpleOCR | Freeware y Comercial | Windows |
| SmartScore | Comercial | Windows |
| Tesseract | Código abierto | Windows, Mac OS X, Linux, OS/2 |

Fuente: (Ordóñez L., 2009).

1.3.5. Text-To-Speech (TTS)

Conocido por sus siglas en inglés (TTS), permite acceder a textos o dispositivos electrónicos para transformarlos en lenguaje hablado por medio de una voz sintética, ya que transforma el texto en voz. Por tanto, permite conocer y manipular dispositivos electrónicos a usuarios con capacidades visuales disminuidas o nulas (Giusti, Lira, Rodríguez Vuan, & Villareal, 2016). También es ampliamente utilizado por personas con discapacidades en el habla y asistencia para múltiples personas con diferentes discapacidades. Por otra parte, es gran utilidad en el funcionamiento de robots de

servicio y a la interacción de humanoides con agentes inteligentes (Izzad, Nursuriati , Noraini, & Norizah , 2015).

1.3.5.1. Funcionamiento

Básicamente TTS consta de 2 partes que son el procesamiento del texto y la generación de voz. En consecuencia, el procesamiento de texto es esencial para una salida del habla sintetizada natural ya que es el encargado de segmentar el texto en una secuencia de unidades de habla fonética o silábica. Las unidades del habla utilizan componentes de generación de voz para producir sonidos del habla. Además, hay que tener en cuenta que la segmentación incorrecta de la sílaba produce un discurso sintetizado antinatural por lo que se califica como una pronunciación inadecuada del habla (Izzad, Nursuriati , Noraini, & Norizah , 2015).

Los módulos más destacados en el sistema de conversión de texto a voz se resumen en la siguiente figura, en donde se aprecia la influencia de algunos sonidos sobre los sonidos adyacentes y el papel que genera los patrones prosódicos (Cávez, 1997).

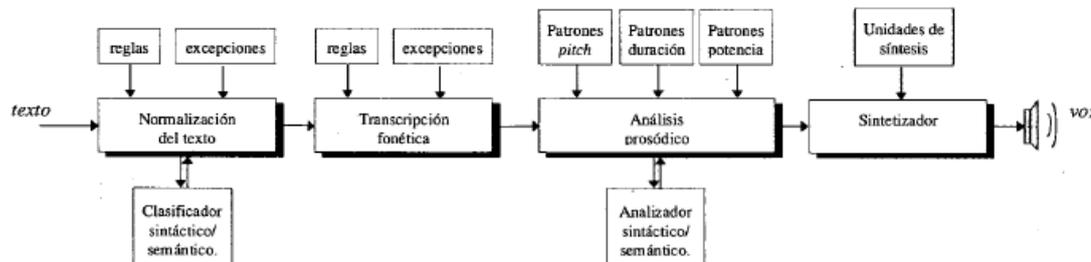


Figura 1.4. Proceso de conversión de texto a voz

Fuente: (Cávez, 1997)

Para detallar el proceso de conversión de texto en voz, en la primera parte del proceso se requiere unos registros de señales orales que se relacionan con las unidades básicas que pueden ser fonemas, los cuales se concatenan conforme es el texto de entrada (Cávez, 1997).

El módulo de la normalización del texto consiste en quitar caracteres de control o de formato, abreviaturas, acrónimos y números para proporcionar de esta manera un texto

escrito ortográficamente al sistema, ya que, existen acrónimos que se leen como palabras una parte y el resto se deletrea como “CD ROM”. Por otra parte, también existe ambigüedad en los signos de puntuación, como por ejemplo el punto puede indicar millares en los números, fin de una frase, inicio de parte decimal, fin de abreviatura, etc. Por lo tanto, para tratar todo lo mencionado se suele tener un analizador para identificar cada componente y verificar si se trata de un acrónimo, abreviatura, palabra ordinaria, etc. Siempre se acepta cierto rango de error o una actualización constante en los módulos (Cávez, 1997).

La silabificación representa el texto de entrada mediante un conjunto de sílabas y la transcripción fonética representa cada conjunto de sílabas mediante fonemas o alófonos. En lenguaje castellano se pueden establecer reglas que relacionen los fonemas que corresponden a las letras, pero en otros idiomas existen excepciones o la utilización de diccionarios con la transcripción fonética de las palabras más probables o todas. Incluso con todas estas herramientas se puede necesitar un análisis sintáctico o semántico para la asignación adecuada a la palabra (Cávez, 1997).

Por otra parte, el módulo que realiza el análisis prosódico es el encargado de determinar las características prosódicas en frecuencia fundamental o también llamada pitch, duración y potencia de cada fonema o alófono, para lograr que la señal de voz tenga continuidad y con características apropiadas al mensaje. La frecuencia fundamental es la frecuencia de vibración de las cuerdas vocales. Por ejemplo, en un patrón melódico de entonación, se produce un incremento de la frecuencia fundamental a partir de la última sílaba acentuada. En cambio, la duración establece el tiempo que tarda cada fonema y cuanto tardan las pausas que aparecen en el mensaje. Además, la potencia depende de cada fonema ya que puede variar de acuerdo con el énfasis dentro del mensaje y de la amplitud de las vocales y algunos sonoros de mayor amplitud. Es por ello que existe una determinada relación entre la frecuencia fundamental y la potencia. La potencia tiene la característica que decae hacia el final de las frases (Cávez, 1997).

Luego, el módulo de síntesis realiza la asignación de fonemas a unidades disponibles después de conocer los fonemas que se quiere concatenar y que características prosódicas, ya que, este módulo se encarga de tomar las unidades básicas, modificarlas y concatenarlas. Finalmente, la última parte del sistema de síntesis, trata de tomar las unidades de síntesis pregrabadas y las modifica de tal manera que posea las características prosódicas deseadas (Cávez, 1997).

1.3.5.2. Aplicaciones

Entre las ventajas que presenta esta tecnología, está en presentar la accesibilidad en forma directa a un texto como: una publicación periódica, diarios o revistas, ya que los nuevos sistemas lectores computarizados generan a un menor costo mayor accesibilidad con una amplia cobertura. Entre las aplicaciones más utilizadas que disponen de convertir texto a voz, se puede nombrar: Acrobat Reader, Natural Read, Aloud, Talk, AudioBookMarker, etc. (Tarín, 2017). Pero, también se consideran desventajas en la tecnología TTS como por ejemplo algunas aplicaciones pueden tener falencias en la calidad de la síntesis, en el alto costo en los equipos, falencias en la rapidez o en la facilidad de operación (Smith Torres, 2009).

1.3.6. Sistema embebido

Cuando se habla de sistema embebido se refiere a un circuito electrónico que realiza operaciones computacionales con el objetivo de cumplir tareas específicas destinadas. Los sistemas embebidos suelen tener recursos limitados con relación a los sistemas de cómputo utilizados en las computadoras de escritorio o en las laptops (Salas Arriarán, 2015).

1.3.6.1. Características

En lo que se trata de la arquitectura, los sistemas embebidos ejecutan instrucciones a una determinada velocidad controlada por una señal de reloj gracias a un microprocesador. La potencia de procesamiento del microprocesador se mide en unidades de MIPS (millones de instrucciones por segundo) y está relacionado con la frecuencia máxima de operación y los recursos internos (periféricos). Por otra parte, los sistemas embebidos generalmente son elaborados en lenguajes como:

ensamblador, ANSI, C, C++ o Basic, pero algunos sistemas embebidos son capaces de ejecutar sistemas operativos limitados, conocidos como RTOS (Real Time Operating Systems), los que permiten ejecutar diversas tareas con prioridades y órdenes de ejecución determinados. Entre los sistemas operativos más conocidos están μ Linux, FreeRTOS, MQX de Freescale, μ C, Windows, etc. (Salas Arriarán, 2015).

Entre las características más resaltadas de los sistemas embebidos está su bajo consumo de energía (Salas Arriarán, 2015). También, se destacan por su reducción de precio, confiabilidad, seguridad (Oleagordia Aguirre, 2012). Adicionalmente, un sistema embebido se orienta a reducir su tamaño, mejorar su desempeño, atender la mayor cantidad de tareas posibles, cuenta con conectividades, cuenta con interfaces de usuario (Cruz, 2013). Además, en la actualidad existen sistemas embebidos en un microcontrolador, que contienen un solo chip y son sistemas computarizados completos, con su interior compuesto por un microprocesador, unidades de memoria (programa y datos), unidades de entrada - salida y los periféricos. Los microcontroladores son de diversas arquitecturas dependiendo de su número de bits, los cuales tienen diferentes marcas y modelos que son diseñados según sea su aplicación (Salas Arriarán, 2015).

En consecuencia, los sistemas embebidos se basan en 3 formas de implementarse, que son: basados en microcontroladores, basados en “Systems on a chip (SOC)”, Híbrido. Los basados en microcontroladores, trata de un sistema autocontenido, que el microprocesador, memoria, soporte y entrada – salida están contenidas dentro de un solo integrado. Los SOC, integran todos los componentes en un chip, tratándose de todos los componentes de un computador y por lo general permiten ejecutar sistemas operativos, prácticamente son una computadora funcional en una sola placa (Grupo MINA, 2017).

1.3.6.2. Aplicaciones

Los sistemas embebidos se encargan de realizar tareas específicas en diferentes áreas, como, por ejemplo, en el campo automotriz pueden controlar el sistema de inyección de gasolina, control de climatización, alarma contra robos, etc. También pueden estar

encargados de diferentes campos como en los equipos médicos, lavadoras, cámaras fotográficas, instrumentación industrial, teléfonos móviles. (Salas Arriarán, 2015). Además, dentro de una fábrica puede controlar el proceso de montaje o producción, puntos de información al ciudadano, sistemas de radar, cajeros automáticos, aplicaciones con comunicaciones embebidas que se orientan a una nueva tecnología con conectividad a internet, nanotecnología y muchas más aplicaciones por descubrir o que estén en desarrollo (Oleagordia Aguirre, 2012).

1.3.6.3. Ejemplos de sistemas embebidos

Tratando de los proyectos de hardware libre más conocidos se tiene a la Raspberry Pi y Arduino, ya que presentan ventajas como tener varias entradas y salidas para el desarrollo de múltiples proyectos. Además, la Raspberry pi, está considerada como un computador de placa reducida, bajo costo y con objetivo de estimular la enseñanza. Por otra parte, se tiene otros dispositivos que también son de hardware libre, es decir que sus esquemas, especificaciones y diseños tienen acceso al público, por lo que entre algunos de los más destacados tenemos: cubieboard, reprop, open compute Project, via openbook, etc. (Roman Bueno & Gonzalez Mantilla, 2016).

El sistema embebido de desarrollo Raspberry Pi nace de “Raspberry Pi Foundation” con su sede en Reino Unido, que consiste en una organización benéfica que trabaja en creaciones digitales que ofrece computadoras a bajo costo y alto rendimiento, capaces de resolver problemas, aprender y divertirse. Hasta el año 2018, la Raspberry Pi 3, es el último producto, con su modelo B+, el cual se indica en la siguiente figura (Fundación Raspberry Pi, 2018).



Figura 1.5. Raspberry Pi 3, modelo B+

Fuente: (Fundación Raspberry Pi, 2018).

Entre las principales características de este modelo se tiene que cuenta con un procesador cuatro núcleos a 1,4 GHz de 64 bits, conectividad LAN inalámbrica de doble banda, Bluetooth 4.2/ BLE, Gigabit Ethernet sobre USB 2.0, SDRAM LPDDR2 de 1GB, 40 pines GPIO extendida, salida estéreo de 4 polos y puerto de video compuesto, entrada de 5V/2.5A en corriente continua (Fundación Raspberry Pi, 2018). En cuanto al consumo de energía, se establece un aproximado de 1.2W en reposo que da un total de 230 mA y a pleno rendimiento un consumo de 1.8W que serían 350 mA, asimilando la alimentación de 5,14V (Velasco, 2017). En la actualidad, presenta un costo aproximado en Estados Unidos de 45 USD en su versión con su kit básico (Fundación Raspberry Pi, 2018).

1.3.6.4. Programación en sistemas embebidos

Para realizar la programación al sistema embebido, se necesita una herramienta de software llamada IDE (Integrate Development Enviroment), la cual puede incluir varios archivos fuente con partes del programa dentro del sistema embebido. Se denomina “firmware” a las instrucciones que especifican el programa en un archivo de texto. Luego de finalizar el programa, se utiliza el software ‘compilador’, que transforma el texto de las instrucciones en código de máquina que está estructurado en

un archivo de números binarios y se envía a la memoria de programa del sistema embebido que se conoce como “Target” (Salas Arriarán, 2015).

Por lo tanto, el lenguaje de programación permite ver en el contexto del programador las aplicaciones que luego estarán en código ejecutable por la máquina. Además, se puede calificar el lenguaje de programación de acuerdo con el nivel de abstracción. Un mayor nivel de abstracción no siempre implica más posibilidades de diseño y menor rendimiento, sino, estamos interponiendo más capas de abstracción entre el hardware y la aplicación que estamos desarrollando. Por ejemplo, en la siguiente tabla se listan algunos de los lenguajes más utilizados, donde se organizan desde el menos abstracto hasta el lenguaje más abstracto, los cuales son (Belmonte , 2014):

Tabla 1.2 Lenguajes de programación en sistemas embebidos.

| LENGUAJE DE PROGRAMACIÓN | EJEMPLOS | ENTIDADES |
|---------------------------------|----------------------------|----------------------------|
| Código de máquina | | Opcodes, hex. |
| Ensamblador | Assembler | Mnemónicos. |
| Lenguaje procedural | C, Pascal, Fortran | Rutinas, módulos. |
| Lenguaje orientado a objetos | C++, Java, Python, etc. | Objetos, clases, mensajes. |
| Framework | C++ STL, Boost, .NET, etc. | Componentes. |
| Herramientas de modelado | UML, etc. | Diagramas. |

Fuente: (Belmonte , 2014).

1.3.7. Lenguaje de programación Python

Python es un lenguaje orientado a objetos de alto nivel que se considera como un lenguaje multi-paradigmas. Python cuenta con múltiples herramientas como estructuras de datos de listas, diccionarios, tuplas, conjuntos, etc. Este lenguaje fue

creado por Guido van Rossum a finales de los 80 (Challenger-Pérez, Díaz-Ricardo, & Becerra-García, 2014). El lenguaje Python es muy potente y accesible, ya que tiene la funcionalidad de que puede funcionar en cualquier plataforma (Walker, 2018). Es un software de código abierto por lo que los usuarios pueden contribuir con documentación y códigos para añadir funciones o corregir errores (Fundación de Software Python, 2018). Su programa hace que el código sea fácil de entender y se escribe en un solo lenguaje, pero suficientemente versátil como para ejecutar en él muchos programas que se desee (Walker, 2018). Además, en la página oficial de Python existe una lista de tutoriales, muchas bibliotecas de Python y recursos en varios idiomas para programadores experimentados (Fundación de Software Python, 2018).

Una de las aplicaciones más utilizadas que fue desarrollada utilizando Python es el motor de búsqueda de Google, la bolsa de New York, sitio web de la NASA, y todas las aplicaciones actuales son solo una fracción de todo lo que puede hacer con Python. Este lenguaje puede ser aplicado a múltiples campos, como es el caso para programas y aplicaciones, inteligencia artificial, juegos, bases de datos, creación web, visión artificial, etc. (Walker, 2018).

1.3.8. OpenCV

OpenCV es desarrollada como una biblioteca libre, robusta y muy eficiente por Intel en 1999, actualmente está entre las bibliotecas más importantes y más utilizadas para visión artificial en el mundo (Del Valle Hernandez, 2018). OpenCV cuenta con un conjunto completo de visión artificial con algoritmos de aprendizaje (Beyeler, 2017). Al ser un software libre, es utilizadas por empresas, universidades y el público en general. OpenCV es escrito originalmente en lenguaje C/C++ y es multiplataforma y se puede ejecutar en muchos sistemas operativos como Linux, Windows, Mac OS X, Android e IOS. Una principal ventaja es que se puede utilizar en diferentes lenguajes de programación, como es el caso de Java, Objective C, Python, etc. (Del Valle Hernandez, 2018).

1.4. Conclusiones

Luego de experimentar la tecnología existente, se puede utilizar los fundamentos teóricos para ampliar el panorama y desarrollar el asistente de lectura con mayor facilidad, por lo cual se puede concluir:

- Se cumplen los objetivos planteados en el capítulo al tener la suficiente información de la tecnología necesaria y el panorama claro del ayudante de lectura que se quiere desarrollar.
- En el sistema OCR es de mayor facilidad el reconocimiento de caracteres cuando se trata de palabras enteras y no solo letras, ya que la información del contexto aporta a relacionar las letras adyacentes al contenido que se está analizando.
- Por otra parte, es difícil conocer el comportamiento exacto de la red neuronal en el sistema OCR o tener un análisis teórico exacto de la red neuronal ya que presenta un comportamiento no lineal y al tener neuronas ocultas no se puede determinar las características del aprendizaje. Pero el algoritmo “back propagation”, es un método muy bueno para entrenar este tipo de redes y aunque no se pueda tener una idea clara de su funcionamiento, se puede confiar que la respuesta del reconocimiento de caracteres va a ser satisfactoria.
- Los módulos texto a voz o conocidos como TTS, tienen cierto grado de error en relación con el texto que se va a interpretar y requieren una constante actualización para minimizar ese error.
- Entre los sistemas operativos disponibles para sistemas embebidos, se concluye que Linux es uno de los más adecuados para su utilización, ya que presta amplia compatibilidad y al ser gratuito se puede implementar sin complicaciones de uso.
- Se concluye también, que el sistema embebido adecuado para la implementación del ayudante de lectura es el módulo Raspberry Pi, gracias a sus indicadas características, comodidad y costo accesible.

- Además, se verifica que el lenguaje de programación Python es acorde a las necesidades planteadas, gracias a su fácil implementación, compatibilidad y gran alcance. Adicionalmente, Python permite utilizar el motor OCR Tesseract idóneo para este sistema. También, la potente biblioteca OpenCV es compatible con el lenguaje Python y dispone de todas las características necesarias para adaptar visión artificial al ayudante de lectura deseado.

CAPÍTULO 2: DISEÑO DEL SOFTWARE ENCARGADO DE ADQUIRIR, SELECCIONAR Y ALMACENAR LAS IMÁGENES.

2.1. Introducción

Luego de haber seleccionado el lenguaje de programación Python para desarrollar el ayudante de lectura, el primer paso es desarrollar el software que se encargue de capturar imagen la seleccione y la almacene para su posterior procesamiento.

Para esto se utiliza la biblioteca OpenCV que la importamos como “cv2” y unas librerías o módulos adicionales que se necesitan importar para realizar el procesamiento de imágenes, como es el caso de: PIL que trata de la biblioteca de imágenes de Python (Python Software Foundation, 2018). Los módulos tratan de algoritmos útiles en un archivo que se puede utilizar o importar a otros módulos o al módulo principal para utilización sin tener que transcribir todo el código de programación nuevamente, sino basta con hacer la llamada de importación inicialmente (Guido van Rossum, 2017). Posteriormente, se hace una breve explicación de lo que trata cada módulo utilizado.

2.2. Módulos utilizados

2.2.1. PIL

Es conocido por sus siglas en inglés “Python Imaging Library”, al utilizar este módulo se integran capacidades de procesamiento para imágenes y admite los formatos más conocidos de archivos proporcionando capacidades muy potentes de gráficos de imágenes (s.a., 2012).

2.2.2. NumPy

Mediante este paquete se reemplaza a los módulos Numeric, Numarray y es encargado de realizar funciones matemáticas rápidas, rutinas para álgebra lineal, generación de números aleatorios, etc. (Athanasias, 2014).

2.2.3. Os

Este módulo es llamado como *interfaces misceláneas del sistema operativo*, y proporciona una forma portátil para ejecutar comandos del sistema (Abedin, 2018).

2.2.4. Imutils

Contiene muchas funciones para el procesamiento de imágenes, las cuales se pueden utilizar para rotación, cambio de tamaño, clasificación de contornos, detección de bordes, etc. (Python Software Foundation, 2018).

2.3. Marco teórico

2.3.1. Imagen digital

Para realizar el procesamiento con imágenes se debe entender como es tratada una imagen en un proceso digital. En consecuencia, una imagen digital es una representación de la realidad que puede ser manipulada por un equipo informático. Las imágenes digitales pueden proceder de fuentes digitales como cámaras o escáner y se pueden clasificar como mapa de bits o imágenes vectoriales. Se dice que las imágenes de mapa de bits están formadas por una matriz de puntos o bits, en donde cada valor de pixel (cada punto que compone la matriz) direccionado por su fila y columna, representa el nivel de gris o de color en el punto. Por lo tanto, el pixel es la unidad mínima de visualización de la imagen digital. En la siguiente figura se indica un ejemplo de la representación de un pixel en una imagen (Departamento de Tecnología IES La Cabrera, 2012).



Figura 2.1. Representación de un pixel en una imagen.

Fuente: (Departamento de Tecnología IES La Cabrera, 2012).

Por otra parte, a las imágenes vectoriales se les conoce también como gráficos orientados a objetos, están conformadas por formas independientes (segmentos, arcos, polígonos, etc.), cada uno con diferentes atributos matemáticos. Presentan una ventaja considerable ya que, ocupan menos espacio de memoria y no pierden su calidad al ser modificados en su amplitud, deformados, rotados, etc. Pero, no pueden almacenar grandes cantidades de memoria y requieren grandes cálculos por el microprocesador, por lo que no es apto para presentar fotografías digitales (Departamento de Tecnología IES La Cabrera, 2012).

Además, se tiene que considerar la calidad de la imagen, y dentro de este tema está la profundidad del color, ya que mediante este parámetro dedicamos una determinada cantidad de bits a almacenar información sobre el color de cada pixel de la imagen. En consecuencia, 1 bit representa 2 colores, 2 bits representa 4 colores, 8 bits representan 256 colores y así sucesivamente. Es por ello por lo que una imagen en representación escala de grises, se obtiene una imagen con muchos tonos de grises -los valores varían según el número de bits de la profundidad del color- en cada pixel, y una imagen monocromática es cuando la profundidad de color es de 1 bit, es decir está formada la imagen por pixeles blancos o negros completamente puros. En la siguiente figura se indica un ejemplo de los modos de color de las imágenes (Departamento de Tecnología IES La Cabrera, 2012).



Figura 2.2. Modos de color de las imágenes.

Fuente: (Departamento de Tecnología IES La Cabrera, 2012).

2.3.2. Resolución espacial

El número de píxeles por fila y por columna en una imagen se le conoce como resolución espacial, por lo tanto, se relaciona con los detalles que se pueden visibilizar en una imagen. Entonces, a mayor resolución espacial menor el área que puede ser representada por un píxel. En consecuencia, con mayor resolución espacial también serán mayores los detalles que se apreciarían en la imagen, es por ellos que se conoce a un píxel como el detalle más pequeño diferenciable dentro de la imagen (Rubio, 2015).

2.3.3. Resolución de niveles

Se conoce también como profundidad del píxel y hace referencia al número de niveles de gris o número de bits que define la intensidad que se ven en una imagen. Por lo tanto, si se tiene n cantidad de bits, el píxel podría tomar valores diferentes en un rango desde cero hasta 2^{n-1} (Rubio, 2015).

2.4. Desarrollo de software

Entonces, para proceder a realizar la captura de imágenes, en la primera etapa se utiliza los módulos: PIL e imutils para realizar las operaciones sobre imágenes; el módulo numpy para los procesos matemáticos sobre imágenes y el módulo os.system para utilizar comandos a través de la línea de comandos. Los pasos adecuados implementados se explican a detalle a continuación.

2.4.1. Captura de la imagen

Para la captura de la imagen, primero se inicia la librería OpenCV asignando a la variable “cap” la captura de la primera cámara conectada al sistema embebido mediante la instrucción: `cap = cv2.VideoCapture(0)`.

Posteriormente, se necesita que la lectura sea constante, por lo que se necesita crear un bucle de lectura repetitivo que en este caso se implementa mediante la instrucción “while()”. Entonces, dentro en bucle se implementa el código que se indica en la siguiente figura.

```

import cv2
from PIL import Image
import numpy as np

cap = cv2 . VideoCapture ( 0 )
while ( True ):
    ret , marco = cap . read ()
    cv2 . imshow ( 'PANTALLA' , marco )
    if cv2 . waitKey ( 1 ) == ord ( 'q' ):
        cv2.imwrite('/home/pi/Documents/TESIS/6/captura1.png',marco)
        break
cap . release ()
cv2 . destroyAllWindows ()

```

Figura 2.3. Idle de Python 3.4, algoritmo para realizar la captura de la imagen de forma manual.

Fuente: Autor.

Por lo tanto, el comando `cap.read()`, hace que se asigne la lectura a la variable inicializada anteriormente y es asignada a la variable “marco”. Ahora para poder visualizar y direccionar el espacio que está captando la cámara, se utiliza el comando “`imshow('PANTALLA',marco)`”, que muestra lo que captura en la variable asignada “marco”, en un espacio de pantalla, en un marco de trabajo nuevo llamado “PANTALLA”. Cabe recalcar que en esta primera etapa del programa el direccionamiento de la cámara hacia el texto y la captura de la imagen es manual. Entonces, se hace una lectura constante del teclado y en el momento de presionar la tecla “q” en el teclado, con el comando de OpenCV “`imwrite('/home/pi/Documents/TESIS/6/captura1.png',marco)`”, se almacena en el directorio asignado una captura de lo que se está monitoreando en la cámara.

2.4.2. Segmentación de la imagen

Para el reconocimiento óptico de caracteres es recomendable tener una imagen segmentada de la porción de imagen que contenga el texto ya que mientras más clara y precisa sea la imagen, mejor es el resultado obtenido. En consecuencia, se implementa el siguiente algoritmo que de todo lo que captura la cámara, dibuje el contorno de la hoja. De esta manera, se tiene identificado que proporción de la toda la

imagen está relacionada con el texto a leer y que porción se va a desechar. En la siguiente figura se indica el código del algoritmo que detecta los bordes de la hoja.

```
import numpy as np
import cv2
import imutils
image = cv2.imread('captura1.png', cv2.IMREAD_GRAYSCALE)
#Carga la imagen anteriormente capturada, pero solo en escala de grises.
ratio = image.shape[0] / 500.0 # Devuelve la dimension de la matriz de la imagen, de nxm, shape(0)=n.
image = imutils.resize(image, height = 500) #Redimensiona la imagen, con la resolución que se dividió para obtener la matriz.
gray = cv2.bilateralFilter(image, 11, 17, 17) #Elimina el ruido de la imagen, tomando en cuenta los pixeles cercanos.
edged = cv2.Canny(gray, 30, 200) #Convierte la imagen en un umbral mínimo y máximo de tonos de color.
_, cnts, _ = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
#Detecta los contornos en una imagen a de color establecido antes.
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
#Ordena descendente de acuerdo al area, los 10 contornos mas grandes.
for c in cnts:
    peri = cv2.arcLength(c, True) #Encuentra el perimetro del contorno detectado.
    aprox = cv2.approxPolyDP(c, 0.01 * peri, True) # Aproxima la curva con una precision relacionada con el perimetro.
    if len(aprox) == 4: # Cuenta el numero de aproximaciones, y si es 4 resulta que es cuadrado.
        cv2.drawContours(image, [aprox], -1, (36, 231, 17), 20, 4)
        #Dibuja el contorno en la imagen original, con borde 20, tipo de línea 4 y color R 36, G 231, B17.
        break #Abandona el bucle.
cv2.imshow("contorno con marco", image) #Vuelve a visualizar la imagen, pero esta vez con el contorno detectado.
cv2.waitKey(0) # espera una tecla para salir del modo de la pantalla.
```

Figura 2.4. Idle de Python 3.4, algoritmo para realizar la detección de contornos.

Fuente: (aprenderpython, 2018).

Por lo tanto, con este código se consigue bordear la porción de imagen interesada para su posterior extracción. Es importante establecer el cambio de color de imagen ya que el comando *findContours* detecta el cambio de color y establece el contorno. Es por ello que no se sabe con exactitud cuántos cambios de colores existen en la imagen y luego se ordenan los 10 contornos más grandes. Claro está que no se puede saber con exactitud si dentro de estos 10 está incluido el contorno de la hoja de papel, por ello se hace una aproximación con relación al perímetro del contorno y si es igual a 4, quiere decir que se trata de un cuadrado. Siendo un borde de un cuadrado no es completa la certeza que se trate de la hoja, pero al tratarse de un fondo de color distinto al del papel, se tienen una muy buena probabilidad de que se trate de la hoja.

De esta manera, se obtiene un resultado favorable ya que podemos hacer un seguimiento del contorno más grande del enfoque de la cámara. Se indica en la siguiente figura un ejemplo del código de segmentación.

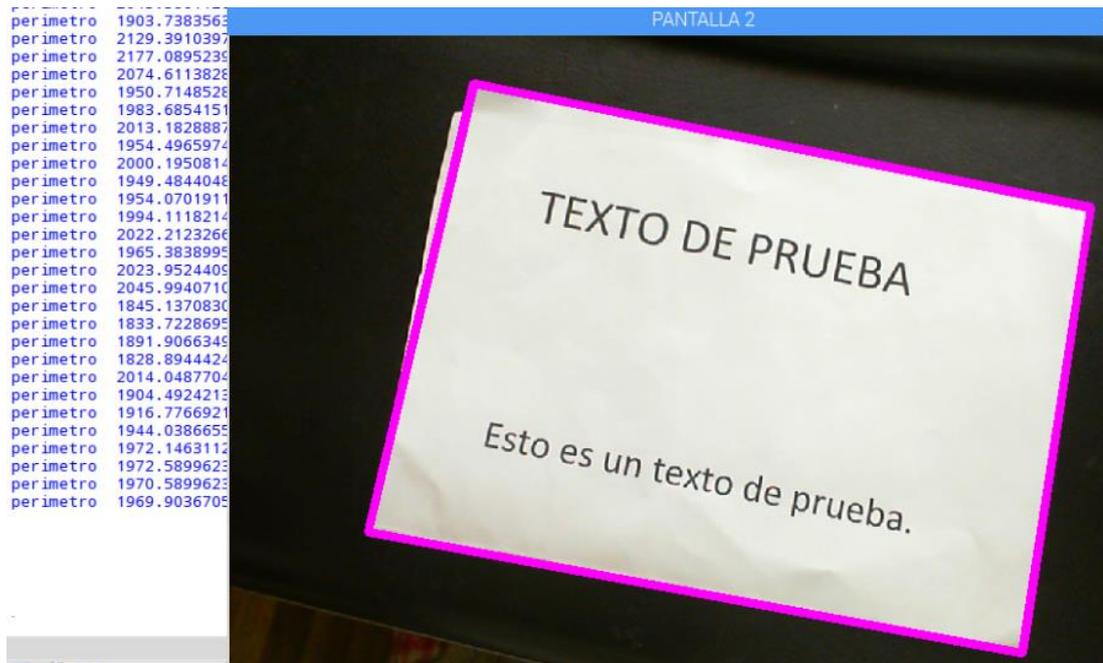


Figura 2.5. Idle de Python 3.4, resultado de la segmentación de la imagen y calibración del perímetro de la imagen.

Fuente: Autor.

En esta sección existe un riesgo de error, ya que al tomar en cuenta 4 puntos para definir que se trate de la hoja, pueden establecerse 4 puntos en muchas figuras hasta llegar a la deseada. Un ejemplo de este error, se indica en la siguiente ilustración. Es por ello por lo que, se toma en cuenta el valor del perímetro del contorno y se establece un límite para saber que se trate de un contorno grande. Entonces, en base a pruebas de valores de perímetros, se establece límites con hojas de tamaño entre A5 o A4. Ya que más grande de estas se tienen q alejar mucho la cámara y el texto es ilegible.

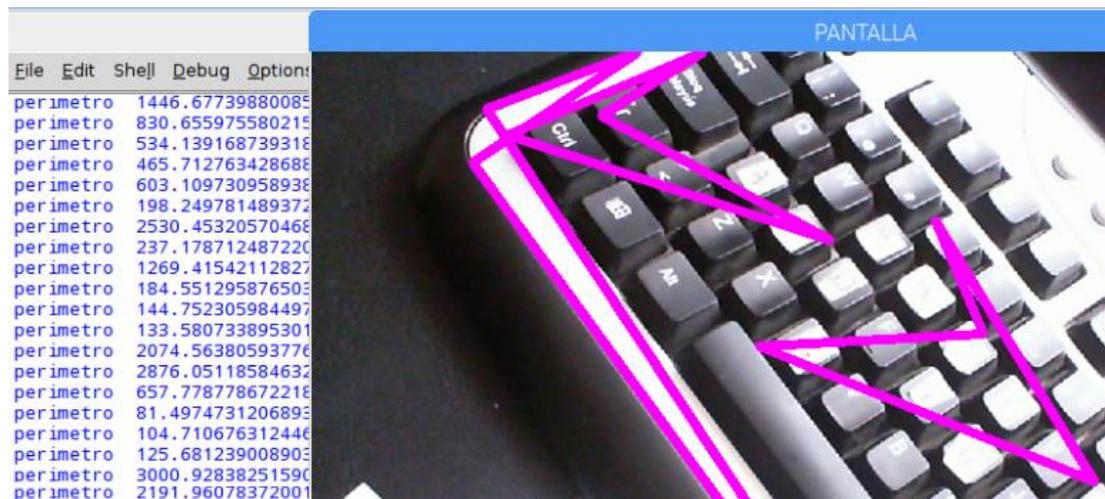


Figura 2.6. Idle de Python 3.4, riesgo de error en la segmentación.

Fuente: Autor.

Como se puede ver en la anterior ilustración, a pesar de que el perímetro es grande, no se trata de la captura del contorno de una hoja de papel. En consecuencia, se implementa una nueva seguridad al código, en donde, se obtienen los valores de los puntos de cada uno de los extremos del contorno y mediante la ecuación de distancia punto a punto de geometría analítica, verificamos que todos los lados del contorno sean mayores a un límite establecido. De esta manera se reduce el riesgo de error, ya que al ser todos sus lados mayores o iguales al resto de los lados es muy probable que se trate de un cuadrado o un rectángulo perteneciente al contorno de la hoja.

A continuación, se indica el código implementado para corregir este error. En donde se establece una distancia de 235, medidos en base a pruebas comprobando la legibilidad de la imagen. Y un perímetro de 1000, lo cual también fue establecido en base a pruebas de campo. Por otra parte, tener el valor del perímetro del contorno, es de mucha ayuda, ya que eso puede ayudar en el momento de direccionar la imagen a la cámara.

```

if (len(approx) == 4): # cuenta el numero de aproximaciones, y si es 4 resulta q es cuadrado.
    #Obtiene las cordenadas de todos los puntos y saca las distancias entre puntos.
    p4=approx[0]; p5=approx[1]; p6=approx[2]; p7=approx[3]
    p0=p4[0]#0
    p1=p5[0]#1
    p2=p6[0]#2
    p3=p7[0]#3
    px0=p0[0]; py0=p0[1]
    px1=p1[0]; py1=p1[1]
    px2=p2[0]; py2=p2[1]
    px3=p3[0]; py3=p3[1]
    dp0p1=sqrt( (px0 - px1)**2 + (py0 - py1)**2)
    dp1p2=sqrt( (px1 - px2)**2 + (py1 - py2)**2)
    dp2p3=sqrt( (px2 - px3)**2 + (py2 - py3)**2)
    dp3p0=sqrt( (px3 - px0)**2 + (py3 - py0)**2)
    if ((dp0p1>distancia) and (dp1p2>distancia) and(dp2p3>distancia) and(dp3p0>distancia)
        and(peri>1000)):
        cv2.drawContours(marco, [approx], -1, (255, 0, 0), 3)

```

Figura 2.7. Idle Python 3.4, corrección de error en la segmentación de la imagen.

Fuente: Autor.

2.4.3. Selección de la imagen

Una vez realizada la segmentación de la imagen, es más sencilla la extracción ya que ya se tiene los puntos coordenados del contorno, por lo que se hace la captura segmentada de la porción de imagen mediante el comando “cv2.imwrite(directorio+'captura1.tif', marco)”. A continuación, se indica un ejemplo de la porción de imagen seleccionada.

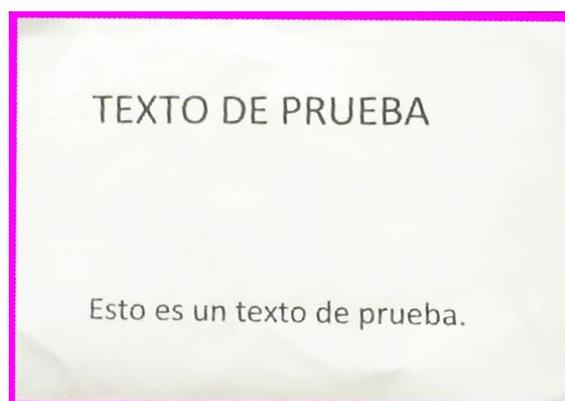


Figura 2.8. Idle Python 3.4, selección de la imagen.

Fuente: Autor.

2.4.4. Almacenamiento de las imágenes

En el apartado anterior se analizó el comando “cv2.imwrite(directorio+'captura1.tif', marco)”, cuando ya se establece la porción de imagen útil. Se implementa un algoritmo que se indica a continuación, encargado de ordenar las imágenes para su posterior análisis.

```
def imagenes():
    global cantidad
    global nuevo
    if ((os.path.exists(directorio+"/imagenes"))and(nuevo==0)):#solo si existe el archivo, lo borra
        os.system("sudo rm -rf "+directorio+"/imagenes")
        nuevo=1
    if not (os.path.exists(directorio+"/imagenes")):
        os.mkdir(directorio+"/imagenes")
    if (cantidad >0):
        cv2.imwrite(directorio+"/captura"+cantidad+".tif",marco)
```

Figura 2.9. Idle Python 3.4, algoritmo para almacenamiento de las imágenes.

Fuente: Autor.

En el código anterior se visualiza que inicialmente se crea un directorio para almacenar las imágenes, pero solo en caso de que no haya sido creado antes. Por otro lado, en caso de que sea un código nuevo, borra el directorio con todas las imágenes que las contenga y en caso de que exista una imagen, la guarda en ese directorio. Se utiliza los comandos de Linux ayudados por el comando “os.system” que permiten escribir como si fuera en la línea de comandos. Hay que considerar aquí que en el almacenamiento de las imágenes es conveniente realizarlos en el formato adecuado.

El formato TIF conocido por sus siglas de “Tagged Image File Format” fue diseñado para almacenar imágenes de mapa de bits de más de 4GB de tamaño comprimido con la particularidad de que se almacenan sin pérdida de calidad ya que, permite almacenar hasta 32 bits por píxel (Vialfa, 2017).

2.5. Conclusiones

- Se concluye que existe mayor precisión en el sistema OCR cuando se procesa un texto en concreto con mayor claridad y precisión, por lo que en la segmentación se debe recortar únicamente el área en interés.
- También es indispensable colocar la hoja del texto que se va a leer en un fondo de un color distinto al de la hoja, ya que la detección de contornos detecta el cambio de colores y permite la posterior segmentación de la imagen.
- Se recomienda tomar en cuenta la calidad de almacenamiento de las imágenes ya que mejora la calidad de la imagen si se guarda con formato TIF, en vista de que este tipo de formato soporta almacenar imágenes de grandes capacidades, incluso mayores a 4GB comprimidos y sin pérdidas.

CAPÍTULO 3: INTERPRETACIÓN DEL TEXTO Y TRADUCCIÓN A VOZ SINÉTICA.

3.1. Introducción

En este capítulo lo que se tratará es traducción de las imágenes que contienen el texto a interpretar a un texto ya digitalizado, y luego a su vez el texto se traducirá a una voz sintética. Este proceso se lleva a cabo gracias a que las imágenes que contienen el texto se encuentran segmentadas y seleccionadas listas para ser interpretadas. Cabe recalcar que para tener la mayor proporción de texto de la imagen reduciendo en lo posible el error en su interpretación, se realizará un procesamiento previo en las imágenes, aplicando filtros que permitan resaltar el texto deseado. Además, se realizarán lecturas para aproximar las condiciones de uso necesarias para el ayudante de lectura.

3.2. Módulos utilizados

3.2.1. Pytesseract

Se lo conoce por su denominación “Python-tesseract” fue desarrollado en los laboratorios de Hewlett-Packard en Greeley (Colorado) en el año de 1989 y consiste en un módulo encargado del reconocimiento óptico de caracteres en Python. Este módulo está movido por el motor “Tesseract-OCR” de Google. También es capaz de leer todos los tipos de imágenes compatibles con la librería de imágenes de Python, como, por ejemplo: png, jpeg, tiff, gif, etc. (Fundación de Software Python, 2018).

Una ventaja muy considerada al momento de utilizar el motor Tesseract es que es motor de OCR libre que se desarrolla actualmente por Google publicado bajo una licencia de Apache. Realiza su reconocimiento primero binarizando la imagen y luego de ser segmentada hace una comparación con la base motora OCR que retorna el resultado en código ASCII. Se considera como uno de los motores más precisos en el reconocimiento de caracteres, por lo que a continuación se indica una comparación entre los motores más utilizados (Garzón Canchignia & Pacheco Gavilánez, 2016).

| | Ocrad | GOCR | Tesseract |
|-----------------------------------|------------------------------------|------------------------------------|----------------------------|
| Precisión en el reconocimiento | Buena | Buena | Excelente |
| Tiempo de Respuesta | Excelente | Mala | Buena |
| Sistemas operativos de desarrollo | Unix | Windows Linux | Windows Linux Mac OS |
| Lenguaje de desarrollo | C++ | C | C y C++ |
| Licencia | GNU Gneral Public License | GNU Gneral Public License | Licencia Apache |
| Formato de imagen de entrada | bmp, pgm, ppm | pnm, pbm, pgm y otros | TIFF |
| Calidad de imagen para análisis | Buena | Excelente | Buena |

Figura 3.1. Comparación en motores OCR

Fuente: (Garzón Canchignia & Pacheco Gavilánez, 2016).

En cuanto a las limitaciones, siempre existen ciertas recomendaciones con respecto al sistema OCR que vale la pena considerar para minimizar el error de respuesta (Geekland, 2015).

1. Se requiere una calidad de la imagen aceptable, ya que si presenta poca resolución o es borrosa el resultado será desagradable.
2. La resolución mínima de la imagen se recomienda que sea superior 300 ppp (puntos por pulgada). Y si la letra es pequeña se recomienda que sean imágenes de resoluciones superiores a 600 ppp.
3. Los resultados pueden ser erróneos en caso de que la tipografía sea poco común. Por ejemplo, en el caso de la manuscrita existen ciertos problemas como la distancia entre caracteres no es contante y muchos más.
4. Los motores y los softwares de reconocimiento no son perfectos, por lo tanto, pueden generar ciertos errores de reconocimiento.
5. Se recomienda un sistema OCR con imágenes a escala de grises o en blanco y negro. Si la imagen no posee esta característica se recomienda un procesamiento previo que también mejoren la calidad de la imagen (Geekland, 2015).

Además, a continuación, se presenta el funcionamiento del motor seleccionado tesseract-OCR.

Primeramente, se analizan los componentes conectados y se almacenan los contornos de los objetos de las imágenes binarias que ingresan. Se agrupan los objetos de contornos similares en amasijos o mezclas desordenadas (Rubio, 2015).

Luego los amasijos se organizan en líneas de texto que se diferencian según un ancho de carácter fijo o variable. En la siguiente figura se explica a que hace referencia el ancho de los caracteres fijos y variables (Rubio, 2015).

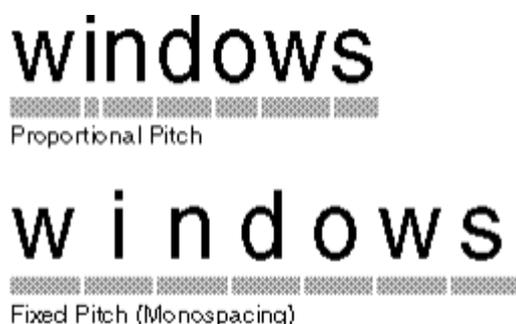


Figura 3.2. Ancho variable de caracteres (proporcional) y ancho fijo.

Fuente: (Rubio, 2015).

Posteriormente, las líneas de textos se dividen en palabras de acuerdo con el espaciado entre caracteres. Si se trata de ancho fijo, cada carácter se introduce en una celda y si es de ancho variable se divide en palabras según valores de espaciados predefinidos (Rubio, 2015).

Luego se produce una fase en la que se intenta reconocer mediante un clasificador palabra por palabra o carácter por carácter del texto. Las que son reconocidas pasan a un clasificador adaptativo o patron de entrenamiento en donde permite más acierto en el avance del texto. Por otra parte, en la segunda fase, se realiza una nueva pasada sobre el texto con todo lo aprendido en el clasificador y se vuelve a tomar desiciones sobre las palabras o caracteres con menor fiabilidad. Finalmente, existen algoritmos que deciden entre caracteres mayúsculas o minúsculas. Y cualquiera que sea el método

OCR, es recomendable realizar un analizador sintáctico que corrija fallos parciales del clasificador, mejore la robustez y la capacidad de acierto del sistema (Rubio, 2015).

3.2.2. Os.system

En este módulo utilizado se puede ejecutar comandos del sistema que se suelen ejecutar desde la línea de comandos, ya que llama al sistema de la función C estándar `system()`, obteniendo las mismas limitaciones (Python Software Foundation, 2018).

3.2.3. Os.system (lame)

La herramienta LAME es un codificador de alta calidad MPEG Audio Layer (mp3) que posee licencia LGPL. Inició en 1998 por Mike Cheng. En la actualidad posee velocidades de bits medias en codificación a mp3. Esta herramienta es compatible con Windows, DOS, GNU/Linux, MacOS X y muchos más (lame.sourceforge, 2018).

3.2.4. Espeak

Es un sintetizador de voz de código abierto escrito en código de lenguaje C a partir de 1955, que proporciona utilidad en muchos idiomas y permite ejecutarse en un programa de línea de comandos de Linux. Utiliza un método denominado “síntesis de formantes” que permite un habla clara en muchos idiomas y en un tamaño pequeño que puede producir una salida de voz en un formato WAV. Además, la velocidad y muchos otros parámetros son configurables pero el habla no es tan natural o suave como otros sintetizadores más grandes que ya por defecto se basan en grabaciones de voces humanas. Espeak puede traducir textos a códigos de fonemas y las herramientas de desarrollo están disponibles para producir y ajustar datos de fonemas. En ventaja, el tamaño compacto de su programa y sus datos con muchos idiomas totalizan alrededor de 2 Mbyte. Se utiliza mucho para escuchar blogs y sitios de noticias (Slashdot Media , 2018).

Un sintetizador por formantes consiste en un modelo estructurado de tracto vocal formado por una fuente y un conjunto de filtros basado en la fonética de producción del habla. Se pueden controlar los valores acústicos que caracterizan cada sonido. La calidad del habla es inferior a otros sistemas ya que se reduce mucho la información almacenada. Se pueden tener configuraciones en serie y paralelo. En serie se relaciona

directamente entre el modelo y las características espectrales y los correlatos articulatorios. Además, no se controla la amplitud ni el ancho de banda ya que lo modifica el siguiente filtro. Presenta dificultad de modelar constantes nasales. Por otra parte, la configuración en paralelo modela espectros complejos con polos y ceros y controla directamente la frecuencia y la amplitud de cada formante. Copia los datos acústicos del análisis del habla real (Llisterri, 2018).

En consecuencia, `system(espeak)` es un comando para un sistema TTS que se efectúa a través de la línea de comandos y es utilizado en ocasiones donde los mensajes se muestren en formato audible en lugar de formato visible. En este caso, la frecuencia de los mensajes son completados en un tiempo prolongado en relación con otros sistemas TTS, ya que son programas de la línea de comandos que no esperan interactuar con los usuarios y se puede tener un mensaje hablado a la vez (Python Software Foundation, 2018).

3.2.5. Gtts

Es una herramienta de la interfaz de línea de comandos y una biblioteca Python conocido por sus siglas en inglés “Google Text-to-Speech” y es utilizada para interactuar directamente con la API de texto a voz de Google Translate por lo que necesita conexión a internet. Su funcionamiento se basa en generar archivos hablados mp3. Trabaja dentro de la licencia “Massachusetts Institute of Technology (MIT)”. Entre los beneficios que presta este módulo se tiene la recuperación automática de idiomas soportados, puede proporcionar correcciones de pronunciación gracias a los preprocesadores de texto, permite la lectura de textos ilimitados gracias a su tokenizador que además mantienen la misma entonación, pronunciación de abreviaturas, decimales y más (Python Software Foundation, 2018).

Su funcionamiento se basa en 3 etapas que son: preprocesador, tokenizer y caja de tokenizador. El preprocesador toma texto, lo modifica corrigiendo en algunos casos la pronunciación, lo repara adecuadamente para la tokenización y devuelve texto. La etapa de tokenizer toma el texto y lo entrega en una lista ordenada de tokens o cadenas que están seleccionadas por símbolos como puntos. Y finalmente la caja de

tokenizador define uno de los casos específicos de “`gts.tokenizer.core.Tokenizer`” y los une para utilizarlos (Python Software Foundation, 2018).

3.2.6. Pygame

Trata de un módulo Python para la biblioteca multimedia SDL con funciones y clases Python que permiten el soporte SDL para salida de audio y video, entrada de teclado, mouse, etc. Posee una licencia LGPL (Python Software Foundation, 2018). Es compatible con Mac, Linux y Windows. Comenzó en el año 2000 en lenguaje C. Es muy utilizado incluso para el desarrollo de juegos ya que posee funcionalidades muy completas y sencillas en la utilización (Zorrilla, 2014).

3.2.7. Time

Proporciona muchas funciones relacionadas con el tiempo. Además, posea herramientas que permiten determinar el tiempo actual del procesador. Y entre otras funciones de las más utilizadas está “`time.sleep(segundos)`”, que permite suspender la ejecución del hilo actual durante un determinado número de segundos establecido (Fundación de Software Python, 2018).

3.3. Marco teórico

Se realiza una breve explicación de los conceptos involucrados en este capítulo, en donde se relacionan al tratamiento de las imágenes para su posterior interpretación.

3.3.1. Filtrado de preprocesado

Existen algunas operaciones que se pueden dar a las imágenes con el objetivo de mejorar ciertos detalles antes de procesarla en el sistema OCR.

3.3.1.1. Transformación de nivel de gris

Se puede aumentar el contraste en una imagen mediante técnicas de tal manera que mejoren ciertas características de la imagen. Es común aplicar a imágenes con falta de iluminación uniforme para equilibrar su iluminación (Navacerrada, 2017).

3.3.1.2. Filtrado Binarización de Otsu

En este tipo de filtrados se establece nuevos valores para los píxeles dependiendo los casos. Por ejemplo, se establece un valor umbral y si el valor del píxel procesado es

mayor a dicho umbral se establece otro valor a ese pixel, el valor establecido puede ser cualquier color, pero, por lo general se utiliza blanco o negro. Dentro de la biblioteca OpenCV la función utilizada es “cv2.threshold”. Los argumentos pueden variar, pero en el filtro umbral simple el primer argumento es la imagen que se va a procesar y debe ser a escala de grises. Luego se establece el valor umbral y luego el valor de pixel que se establecerá (opencv dev team, 2014).

Entonces, para la Binarización de Otsu se establece un valor umbral arbitrario que el mismo método va probando el error y lo actualiza automáticamente. En el caso de una imagen bimodal (su histograma contiene 2 picos) se puede aproximar el valor de umbral en medio de los picos y precisamente eso hace este tipo de filtrado. El algoritmo de Otsu intenta auto calcular el umbral que minimice la varianza ponderada dentro de la clase y encuentra el valor que esté entre dos picos. Es por ello que cuando la imagen no es bimodal, existe mayor error en el cálculo del valor umbral y la Binarización no es precisa. En la siguiente figura se indica un ejemplo de cómo sería un filtrado si se le asigna un umbral de 127 y luego el filtrado auto calcula y corrige el umbral (opencv dev team, 2014).

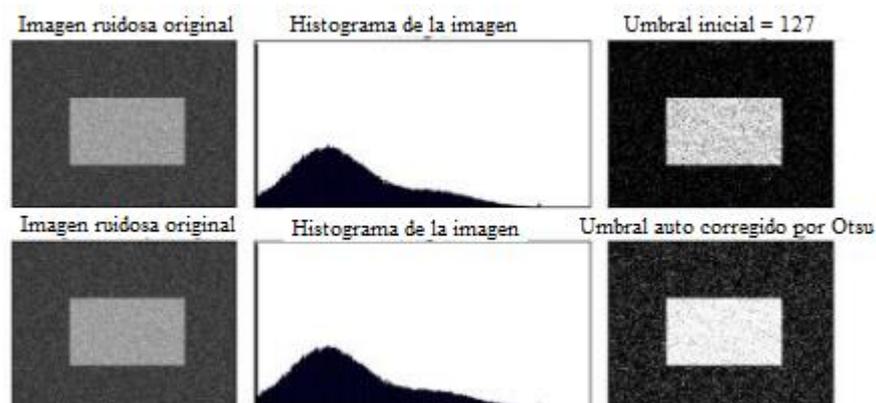


Figura 3.3. Binarización de Otsu.

Fuente: (opencv dev team, 2014).

3.3.2. Codificación

Las secuencias de caracteres son legibles únicamente por los humanos, pero para que puedan ser interpretados por las máquinas y serán almacenados se deben pasar a una secuencia de bytes mediante una codificación (Singh, 2017).



Figura 3.4. Codificación de caracteres.

Fuente: Autor.

Existe diferentes formas de codificación como por ejemplo algunos formatos que permiten representar audios, imágenes, textos y muchos otros archivos en bytes, como por ejemplo: PNG, JPEG, MP3, WAV, ASCII, UTF-8, etc. (Singh, 2017). En el programa Python se trata con ASCII y Unicode para las codificaciones pero, propiamente Unicode es la cadena predeterminada, ya que gracias a esta se pueden incluir todos los caracteres en todos los idiomas. Las cadenas se pueden crear con el simple hecho de encerrar los caracteres dentro de comillas simples, dobles o triples (Programiz, s.f.).

La cadena Unicode trata de una secuencia de puntos de código compuesta por números del 0 al 0x10FFFF y existe una tabla de conversión asignada para cada carácter. En su representación directa se signa una matriz de enteros de 32 bits, lo cual es un desperdicio de espacios en la mayoría de textos, ya que, cada carácter ocuparía 4 bytes y en ASCII ocuparía solo 1 byte, lo cual minimizaría en la cuarta parte el uso del disco duro de un ordenador. En consecuencia, generalmente se elige utilizar UTF-8 como codificación. UTF-8 no maneja todos los caracteres de Unicode sino si el punto de código es menor a 128 cada byte es el mismo valor caso contrario expresa un error. La codificación Latin-1 también es muy utilizada y su utilización es similar ya que utiliza los mismos puntos de código Unicode de 0 a 255 y superior a eso no se puede codificar (Python Software Foundation, 2018). En la siguiente figura se indica un ejemplo de cómo será la codificación de la cadena de caracteres “PYTHON”.

| | | | | | | |
|------|----|----|----|----|----|---|
| P | y | t | h | o | n | Cadena de caracteres Codificación UTF-8 y Latin-1 Bytes utilizados |
| 0x50 | 79 | 74 | 68 | 6f | 6e | |
| 0 | 1 | 2 | 3 | 4 | 5 | |

Figura 3.5. Codificación UTF-8 y Latin-1.

Fuente: Autor.

Dentro de este tema es importante conocer que las variables en el programa Python al ser un espacio de memoria reservado con capacidad de cambiar, Python dispone de precisión ilimitada, es decir, se asignan al menos 32 bits para cada variable y en caso de requerir mayor capacidad en una variable es asignada automáticamente limitando únicamente por la memoria disponible del dispositivo donde se ejecute el programa (Instituto de Astrofísica de Andalucía, s.f.).

3.4. Desarrollo del software

Dentro de este capítulo se realizará el procesado de las imágenes ya obtenidas, de tal manera que se pueda obtener una mejor imagen de la cual se pueda extraer la información deseada y posteriormente obtener una salida audible del texto.

3.4.1. Filtrado de la imagen

Para mejorar la calidad de la imagen se aplican una serie de correcciones, inicialmente se aplica la conversión de la imagen a escala de grises mediante el comando “gray = cv2.imread('imagen.tif', cv2.IMREAD_GRAYSCALE), ya que lee la imagen y la transforma en un mismo comando.

Una vez que se tiene la imagen en escala de grises se aplican los filtros deseados, y en este caso se inicia con un filtrado de los más óptimos perteneciente a la librería OpenCV que se denomina filtrado gaussiano. En este caso se aplica el siguiente comando “th3=cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY, 11,2)”, en el cual se establece el color blanco por encima del valor umbral y se indica el filtrado en la siguiente figura.



Figura 3.6. Idle de Python 3.4, resultado del Filtrado Gausseano.

Fuente: Autor.

Al ver que existe la generación de muchos puntos negros alrededor del texto, se implementan operaciones morfológicas que eliminen estos inconvenientes. Por lo tanto, se genera una matriz de unos que esblencarán los valores de ancho y alto de la proporción de operación que se aplica a la imagen mediante el comando “`kernel2 = np.ones ((2,2), np.uint8)`”. En este caso se trata de una matriz pequeña de 2x2 ya que se trata de textos de letras pequeñas. Entonces, se realiza 1 iteración en dilatación para reducir los puntos negros y para recuperar lo perdido otra iteración de erosión mediante los comandos “`cv2.dilate (th3, kernel2, iterations = 1)`” y “`cv2.erode (dilation, kernel2, iterations = 1)`”. Se aprecia que la imagen mejora, pero no se soluciona del todo, ya que existen puntos negros del ancho de las letras y al aumentar la capacidad de las operaciones se perdería mucha información irrecuperable como se indica a continuación.

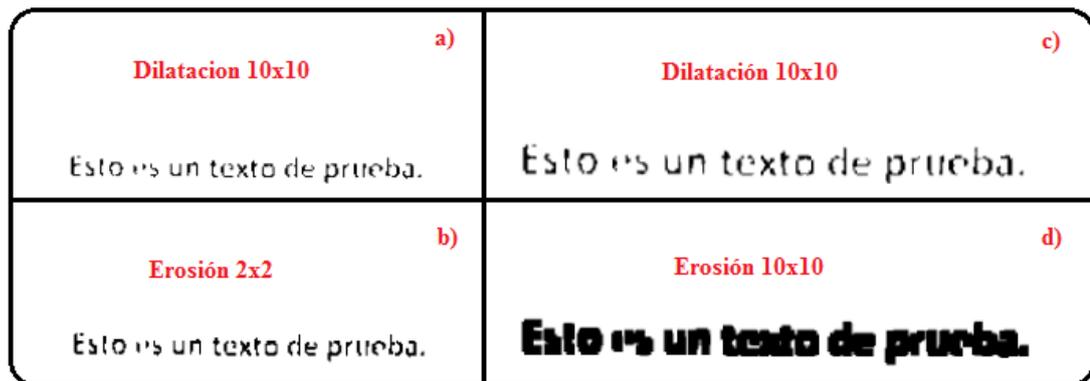


Figura 3.7. Idle de Python 3.4, resultado de operaciones morfológicas.

Fuente: Autor.

En la ilustración anterior se puede ver los resultados de las operaciones morfológicas, en donde las imágenes *a* y *b*, fueron tratadas a una resolución de 500 y las imágenes *c* y *d*, fueron ampliadas a 1000. En ambos casos se puede visualizar que en vista de ser las letras pequeñas y unidas tienden a asemejarse a los puntos negros y al eliminarlos también se pierde cantidad en las letras. Se intentaron con varios tipos de matrices de diferentes tamaños, pero no se obtuvo un resultado favorable. En la imagen *d*, se exagera la matriz con el fin de indicar que, si ya se perdió parte de las letras, no importa que tan grande sea la matriz de erosión que igual se dificulta recuperar el texto. Es por ello que se intenta con otros tipos de filtros.

En consecuencia, se implementa la Binarización de Otsu mediante el comando `cv2.threshold(gray,0,255,cv2.THRESH_BINARY/cv2 . THRESH_OTSU)`, el cual se establece de igual manera con la imagen en escala de grises y refleja el siguiente resultado.

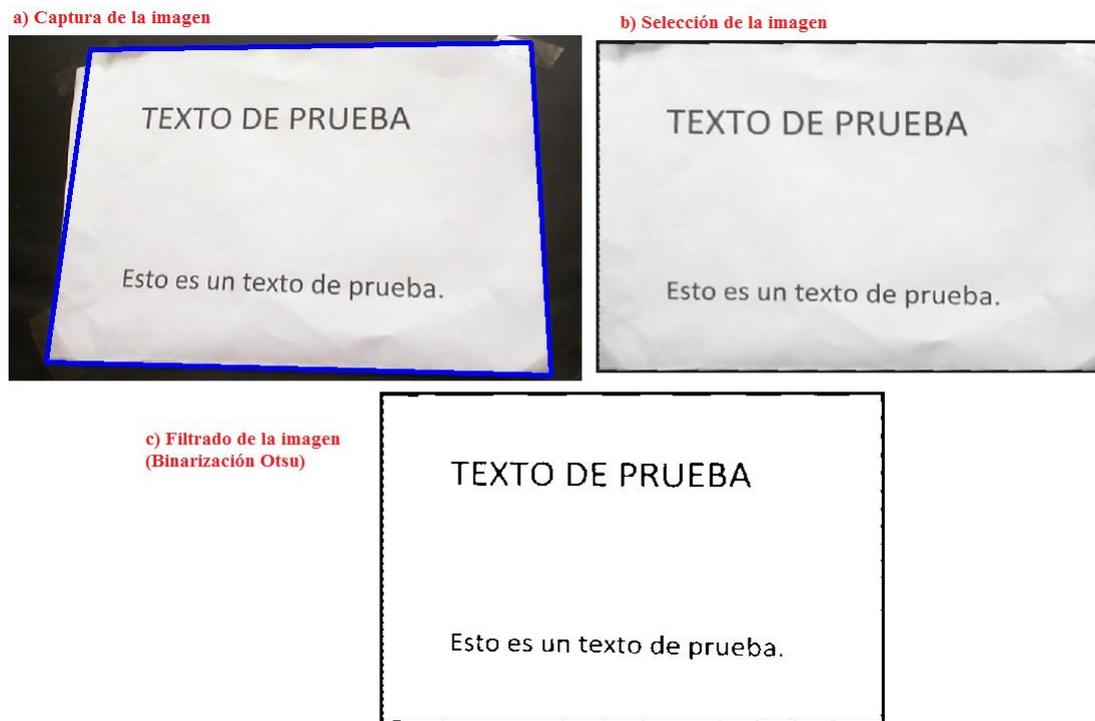
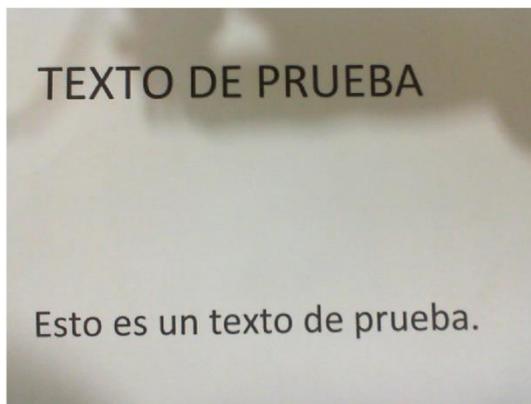


Figura 3.8. Idle de Python 3.4, preprocesado de la imagen.

Fuente: Autor.

En la ilustración anterior, se verifica que el filtro nos proporciona un resultado muy satisfactorio, ya que la imagen filtrada ya no presenta puntos y es completamente legible. Posteriormente, se realizan varias pruebas con diferentes tipos de textos y hojas y se encuentran algunas complicaciones. Por ejemplo, si la imagen capturada contiene una sección con sombra, pero la sombra no es constante en toda la imagen sino solo una porción, el umbral que auto calcula la Binarización Otsu encuentra un inconveniente y se produce un error en el filtrado como se puede indicar a continuación.

a) Imagen capturada con una sección de sombra



b) Binarización Otsu de imagen con sombra

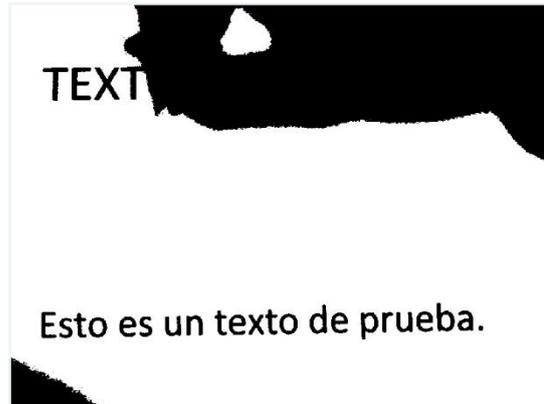


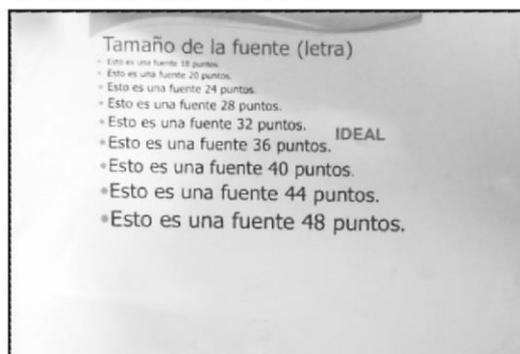
Figura 3.9. Idle de Python 3.4, error en el filtrado a causa de captura errónea.

Fuente: Autor.

Por lo tanto, se requiere que la imagen sea tomada con el texto sobre un fondo oscuro y a un solo tono de luz, ya que incluso puede tener sombra toda la imagen, pero no solo una porción.

Otro inconveniente se presenta en el filtrado cuando los textos son de tamaño pequeño, es decir menores a 36 puntos y de ancho delgado, ya que al momento de filtrar se pueden encontrar pixeles con rangos similares a los no deseados dentro de las letras y por lo tanto se eliminarán. Por lo general esto sucede con los bordes de las letras como es indicado a continuación.

a) Selección de la imagen de textos variables.



b) Filtrado de la imagen de textos variables.

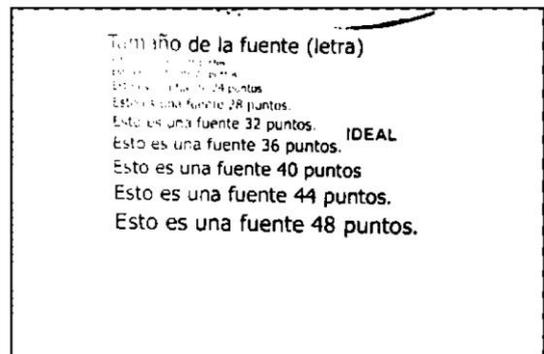


Figura 3.10. Idle de Python 3.4, limitaciones de tipo de textos.

Fuente: Autor.

En consecuencia, se limita a leer textos mayores a 36 puntos de tamaño, ya que textos más pequeños se pierden las características de las letras haciendo ilegible los textos.

3.4.2. Obtención del texto de la imagen

Para la obtención del texto que contiene las imágenes, se utiliza la biblioteca *Pytesseract*, la cual devuelve una cadena de caracteres con el texto que contiene la imagen que se le asigna como parámetro. Por ello, se utiliza el comando `t1=pytesseract.image_to_string(Image.open(directorio+'/filtrado2.tif'),lang='spa')`, por lo tanto, en `t1` queda almacenado el texto en tipo *string*. Posteriormente, se realiza una conversión de tipo de texto para evitar caracteres error y para optimizar recursos mediante el comando `t1=str(t1.encode("utf-8").decode('latin-1'))`.

En este caso, se realizan varias pruebas en diferentes tipos de textos y diferentes tamaños de hojas y se llega a la conclusión que se necesita a más de textos de tamaño grande, acercar considerablemente la imagen a la cámara. Esto se debe, ya que textos grandes superiores a 36 puntos pueden considerarse pequeños con distancias lejanas. Por ello, se realizan pruebas y en base a una investigación experimental se detallan ciertos parámetros adecuados para la toma de las imágenes.

La siguiente imagen fue tomada a una distancia aproximada de 70 cm, con una cámara HD de 720p (720 líneas horizontales de resolución de pantalla de barrido progresivo) y 30 fotogramas por segundo a una hoja de tamaño A4. Cabe recalcar, que la distancia es la más corta para capturar toda la hoja A4. Al momento de realizar la conversión a texto, se determina un error en la lectura, ya que genera un texto ilegible apesar de que se indica un texto aproximadamente grande y dentro de las características legibles elegidas anteriormente.

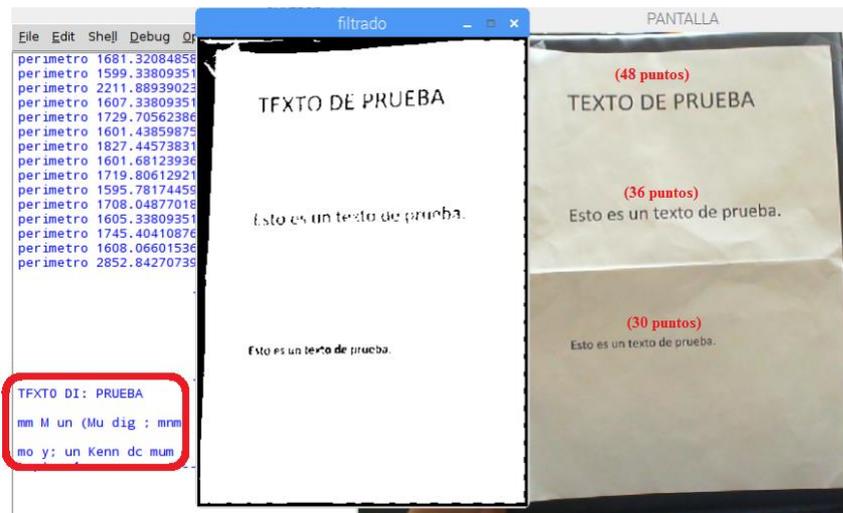


Figura 3.11. Idle de Python 3.4, obtención de texto de una hoja tamaño A4.

Fuente: Autor.

En consecuencia, se continúan realizando pruebas en diferentes tipos de textos y a diferentes distancias y se obtiene lo indicado a continuación.

En la siguiente ilustración se realiza una prueba con textos blancos en fondos de colores, y se verifica el porcentaje de lectura. Se puede constatar que en el filtrado se obtiene un buen resultado, pero a pesar de eso en el momento de la interpretación no se puede obtener el texto en su totalidad, por lo que, se recomienda que sean textos de color negro en un fondo blanco para minimizar el error.

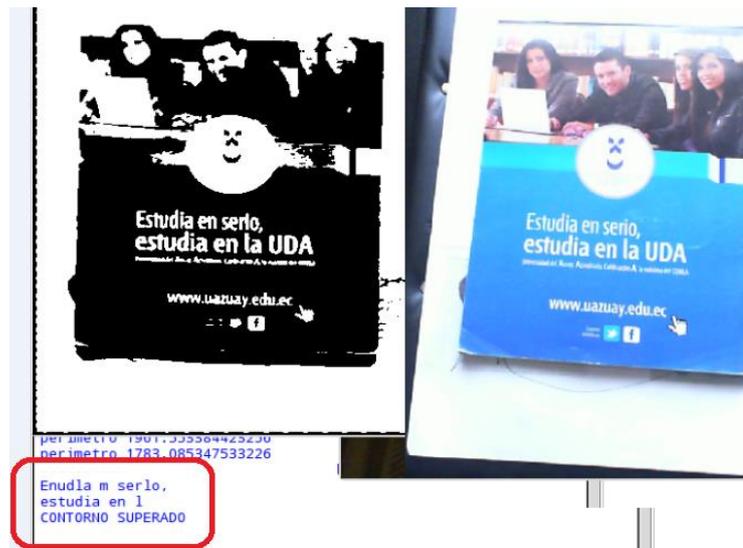


Figura 3.12. Idle de Python, textos con fondos a color.

Fuente: Autor.

Posteriormente, se establecen diferentes tomas de imagen, al ver que se necesita alejar más la cámara en hojas de tamaño A4, se reduce la hoja una de tamaño A5, con el propósito de poder acercar más la cámara y que se pueda capturar toda la hoja. Y se obtiene el siguiente resultado.

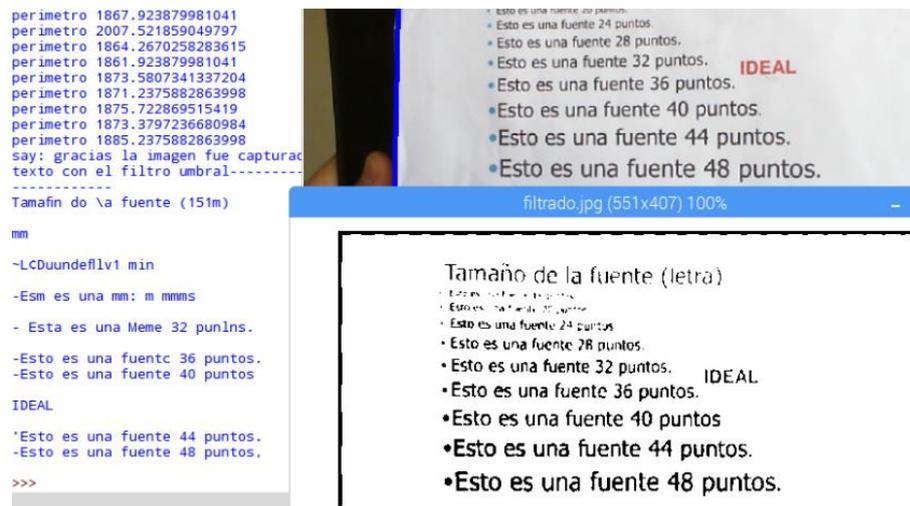


Figura 3.13. Idle de Python 3.4, texto en hoha de tamaño A5.

Fuente: Autor.

La imagen fue tomada a una distancia aproximada de 40 cm de la cámara, por lo que con esta distancia y la resolución establecida de 500 pixeles en el software se logra capturar perfectamente todo el texto. Se recomienda establecer textos de color negro con letras de color blanca de tamaños iguales o superiores a 36 puntos en fondo blanco, ya que se verifica que el error se minimiza siguiendo estas recomendaciones.

3.4.3. Generación de la voz a través del sistema TTS

Como ya se mencionó el módulo utilizado es *espeak* de la línea de comandos. Por ello se crea una función en donde crea una carpeta con cada texto de cada página examinada que posteriormente se van a reproducir. En la siguiente figura se indica el algoritmo utilizado.

```
def voz(texto):
    global cantidad
    if not (os.path.exists(directorio+"/audios")):
        os.mkdir(directorio+"/audios")
    if (cantidad > 0):
        os.system("espeak -v es-la+f2 -k 10 -s 120 \""+texto+"\" --stdout > "+directorio+"/audios/temporal"+str(cantidad)+".wav")
        os.system("lame "+directorio+"/audios/temporal"+str(cantidad)+".wav "+directorio+"/audios/temporal"+str(cantidad)+".mp3")
        cantidad=len(glob.glob(directorio+"/audios/*mp3"))
```

Figura 3.14. Idle de Python 3.4, algoritmo de generación de voz.

Fuente: Autor.

Como se puede visualizar se crea el directorio contenedor de los audios solo si no existe. Luego se genera cada audio de una manera ordenada conforme se lea página por página, ya que, cada imagen que se captura genera un nuevo audio. En este caso se trabaja con muchas de las características se pueden manipular del módulo *espeak*. Por ejemplo, *-v* hace referencia a la voz en donde se establece el idioma español latino con una voz femenina. Además, *-s* tiene que ver con la velocidad de lectura, ya que se establece ese valor de 120 palabras mediante pruebas para no generar una lectura muy rápida y q sea entendible, a pesar de que la velocidad de lectura visual como se vio en el capítulo 1 está entre las 200 palabras por minuto. En este caso, la voz es un tanto robótica, pero es el mejor recurso, ya que, se puede utilizar sin conexión a internet.

Otro inconveniente que genera este módulo es que devuelve audios en formatos WAV y se necesita que esté en formato mp3, por lo que es necesario inmediatamente de

generados convertirlos para su reproducción sin problema. Por ello, se utiliza el comando `os.system("lame")`.

Como el módulo `gtts`, posee múltiples ventajas, se realiza las voces del menú y la interacción con el usuario al ser una voz más natural y al tener la posibilidad de generar audios pregrabados. Por lo tanto, mediante los comandos `tts = gTTS(text=texto,lang='es',slow=False)` y `tts.save(directorio+"/menu/"+numero+".mp3")` se van generando todos los textos necesarios para la interacción con el usuario y en momento del programa que se requieran solo se los reproduce.

En cuanto al espacio requerido, podemos verificar con los parámetros del sistema que son archivos muy livianos. Para tener una idea un audio de 20 palabras ocupa un espacio de memoria de 8 kb.

Para reproducir los audios generados se utiliza una sola biblioteca encargada de reproducir archivos en formato mp3, se trata de la biblioteca `pygame`. Esta biblioteca tiene una gran ventaja de que permite reproducir audios de manera paralela a otros procesos y no suspende el monitoreo de la cámara mientras reproduce. Posteriormente, se indica el algoritmo utilizado para la reproducción.

```
def reproducir():
    global cantidad
    pygame.init()
    if (len(glob.glob(directorio+"/audios/*.mp3"))>0):
        for i in range(1,cantidad+1):
            p.set_volume(v)
            p.load(directorio+"/audios/temporal"+str(i)+".mp3")
            p.play(loops=0, start=0.0)
            time.sleep(1)#Tiempo que se demora para separar hojas o textos leidos.
        cantidad=0
```

Figura 3.15. Idle de Python 3.4, algoritmo de reproducción de audios.

Fuente: Autor.

También se aprovechan las propiedades de la biblioteca, ya que se establece una variable para poder manipular el volumen en cualquier momento del programa. Además, mediante las funciones se puede detener o reanudar cualquier audio en cualquier momento.

3.5. Conclusiones

- Existen muchos tipos de filtrados para las imágenes y muy buenos, como por ejemplo el filtrado de OpenCV gaussiano, pero deja puntos muy pequeños en los contornos que provocan errores muy serios en la lectura OCR. Además, aplicar operaciones morfológicas para eliminar las imperfecciones es recomendable solo en el caso de tener una letra con espaciado grande, ya que caso contrario con la dilatación que elimina los negros, deja muy delgadas las letras y no se puede solucionar ni siquiera con la erosión ya que si son muy delgadas las letras se pierde la información. Por lo tanto, se concluye que el mejor método para esta aplicación es el filtrado de Otsu, ya que establece un nivel umbral de acuerdo a las necesidades de cada imagen.
- Se recomienda que las condiciones de luz sean constantes en la captura de la imagen, ya que al tener porciones de sombra dentro de la imagen genera error en el filtrado dando una salida al filtro con manchas en la sección de sombra.
- Se encuentra una limitante a los textos, ya que no pueden ser de tamaños menores a 36 puntos, caso contrario de ser menores se pierden las características de las letras en el filtrado y se obtienen textos ilegibles.
- Se recomienda la utilización del módulo *espeak* para el sistema TTS, a pesar de que *gtts* dispone de una voz más natural, ya que *espeak* no requiere conexión a internet y puede ser utilizado en cualquier momento.
- Imagen por analizar debe tener ciertas características para mejorar la precisión de reconocimiento, por ejemplo, debe tener una buena resolución en la cámara, como es el caso de una imagen pequeña que sea mayor a 600 ppp. Además, cuando la tipografía no es muy común, puede extenderse el riesgo de un error

en el reconocimiento. También, siempre hay q recordar que los sistemas OCR no son perfectos y puede existir un error en la interpretación.

- Se concluye que la variable utilizada que contiene los caracteres leído y a pronunciar en el programa es de tipo *string* que cada caracter ocupa 1 byte de memoria y tiene una capacidad limitada únicamente por la capacidad del dispositivo donde se ejecuta el programa. En este caso al tratarse de un sistema embebido raspberry pi 3 de 64 bits con 1GB de RAM y una memoria externa de 32 GB se define que dispone de una capacidad superior a la necesaria para almacenar los caracteres leídos de una hoja de papel entre tamaño A5 o A4 con el tipo de letra seleccionada de ancho superior a los 30 puntos. Por lo tanto, la única limitación se da por la cantidad de caracteres que ingresen en la hoja a leer, ya que en cada hoja por separado se genera un archivo independiente de audio que al final se reproduce uno por uno.

CAPÍTULO 4: COMPROBACIÓN E INTEGRACIÓN DE LOS SISTEMAS IMPLEMENTADOS.

4.1. Introducción

Este capítulo trata de la integración de todos los sistemas desarrollados anteriormente y sus respectivas pruebas de campo. Además, se desarrollará el sistema de control que permita la interfaz con el usuario. Finalmente se realiza un resumen de los alcances y las limitaciones que presenta el ayudante de lectura.

4.2. Módulos utilizados

4.2.1. GPIO

Es un módulo encargado de proporcionar el acceso a los pines GPIO (“General Purpose Input/Output”, Entrada/Salida de Propósito General) de la raspberry. Los pines se encuentran a lo largo del borde superior del tablero y cuentan con 40 pines que se pueden designar mediante software para una amplia variedad de propósitos. Hay que tener en cuenta que tanto la tensión de entrada como la de salida son tolerantes a solo 3.3 VDC. Algunos ejemplos de las variedades de configuración de los pines son: PWM, SPI, I2C, etc. (Fundación Raspberry Pi, 2018).

4.3. Marco teórico

4.3.1. Fuentes de textos

Existen muchos tipos de fuentes de texto, entre las cuales las que tienen símbolos y las que no. Las que no tienen ningún símbolo se les conoce también como no escalable o fuentes de trazos y generalmente son utilizadas en las impresoras. Por otra parte, entre las que contienen símbolos se encuentra *Symbol*, este tipo de fuente incluye el alfabeto griego, símbolos de los palos de la baraja, símbolos químicos, matemáticos, comerciales, etc. (González & Florez López, s.f.)

Por otra parte, el tamaño de las fuentes es un parámetro que vale la pena considerar. Puesto que, los tamaños dependen mucho del tipo de fuente que se va a utilizar. En las fuentes existe una unidad de medida para los caracteres que se conoce como *puntos*. Dentro del cual, en tipografía un punto mide aproximadamente 0,035 cm. En

consecuencia, en una pulgada existen 72 puntos (72ppp) y por lo tanto 28,3 puntos es equivalente a un centímetro (González & Florez López, s.f.).

4.4. Desarrollo del software

4.4.1. Integración del sistema

En esta etapa se realiza una captura de la imagen de manera automática, puesto que, se realiza la integración del sistema y todas las etapas ya desarrolladas deben actuar en secuencia. Dentro del capítulo 2 se desarrolló la obtención de las imágenes en donde la captura se la hacía manualmente a través de la *tecla q*. Ahora como es un proceso automático y con la ayuda del perímetro del contorno visto en la sección 2.4.2, se realiza un seguimiento de la imagen, en donde se cuenta un número de coincidencias percatándose de que la imagen no se haya movido y se establece la captura automática. El algoritmo utilizado se indica a continuación.

```

if (peri<1450):
    print(" Acerque un poco el texto. ")
    vmenu("12")
else:
    if contador==0:
        print(" Mantenga el texto en ese lugar por un momento. ")
        vmenu("13")
        contador=contador+1

```

Figura 4.1. Idle de Python 3.4, algoritmo de acercamiento al texto.

Fuente: Autor.

En el algoritmo anterior se puede visualizar que se establece un valor de 1450 para el perímetro, tomado en base a pruebas de visualización de textos en hojas de tamaño A5 con la precaución que se capture todo el texto. La distancia de captura está entre los 35 cm a 50 cm. Entonces, si todavía no se enfoca toda la hoja no llegará al valor de 1450 y se reproduce una aleta audible indicando que se debe acercar más. En caso de tener una imagen adecuada se establece un contador de coincidencias en el cual solo la primera vez indicara que esta correcta la posición y que lo mantenga ahí hasta capturar la imagen.

En este caso se determina un valor de 10 coincidencias para la captura automática de la imagen, lo que asegura que la imagen permaneció enfocada, tal y como se indica a continuación.

```
if contador==10:#SI EXISTE 10 COINCIDENCIAS, TOMA LA CAPTURA PARA LEER.
    print(" Gracias la imagen fue capturada, espere un momento. ")
    vmenu("17")
    contador=0
#-----
#codigo de captura de la imagen para leer
cv2.imwrite(directorio+'captura1.tif',marco) #Realiza la captura automática de la imagen.
captura = cv2.imread(directorio+'captura1.tif', cv2.IMREAD_GRAYSCALE)# Lee la imagen.
```

Figura 4.2. Idle de Python, algoritmo de captura automática.

Fuente: Autor.

En vista de tener como referencia el perímetro del contorno se puede saber que tan aproximada está la imagen de la cámara. Por lo tanto, se establecen varias ayudas audibles para direccionar la cámara hacia la imagen como se indica en el siguiente código que se establece en caso contrario de no encontrar ningún perímetro que satisfaga con las condiciones establecidas en la sección 2.4.2.

```
lejos=lejos+1
if lejos>3:
    contador=0
if ((lejos==4)and(decir>0)):
    print(" Se ha perdido la imagen.")
    vmenu("14")
if lejos==40: #17s
    print(" El texto se encuentra fuera del alcance de la cámara. ")
    vmenu("15")
if lejos==70: #26s
    print(" Por favor direccionar el texto a la cámara. ")
    vmenu("16")
```

Figura 4.3. Idle de Python, algoritmo de ayudas audibles al seguimiento de la imagen.

Fuente: Autor.

Como se puede observar, por cada coincidencia que determine que no se está enfocado a la hoja, una variable denominada *lejos* aumenta y dependiendo de la variación se produce una alerta diferente que ayudará al usuario a enfocar la imagen.

Por lo tanto, luego de obtener la imagen, se puede continuar con los siguientes procesos ya desarrollados anteriormente, en donde se establecen en secuencia dando una salida audible del texto capturado. Se realiza un resumen de este proceso en la siguiente figura, en donde se establece una integración de los sistemas una vez capturada la imagen.

```

captura = cv2.imread(directorio+'captura1.tif', cv2.IMREAD_GRAYSCALE)
#-----
#filtrado
th3 = cv2.threshold(warped, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
cv2.imwrite(directorio+'filtrado.tif', th3)
#-----
#libera recursos
cap.release()
cv2.destroyAllWindows()
#-----
#ocr
t=pytesseract.image_to_string(Image.open(directorio+'filtrado.tif'), lang='spa')
print(t)
#-----
#tts
vmenu(" El texto es: ") #Carga audios pregrabados.
voz(t) # Lee el texto OCR.
vmenu(" El texto ha terminado.") #Carga audios pregrabados.

```

Figura 4.4. Idle de Python, algoritmo de integración de los sistemas.

Fuente: Autor.

Posteriormente, con el sistema integrado se realizan varias pruebas de campo sobre textos, como es el caso que se presenta a continuación.

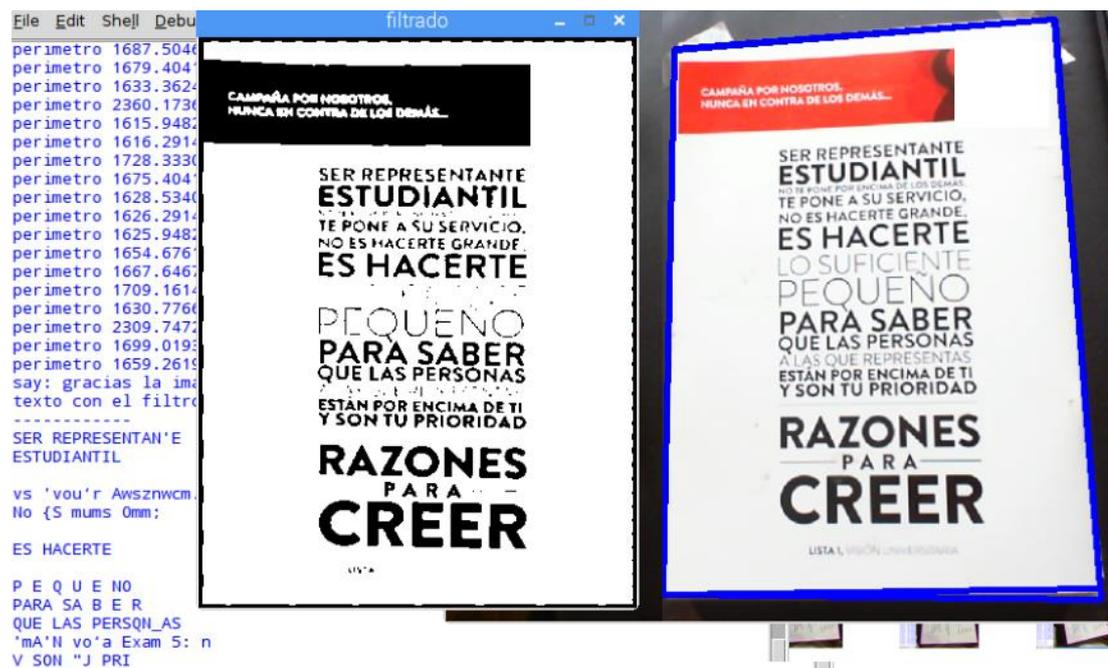


Figura 4.5. Idle de Python 3.4, pruebas sobre textos de diferentes anchos.

Fuente: Autor.

En este caso, se tienen textos sobre diferentes fondos, diferentes tamaños y diferentes anchos. Lo que se verifica que presenta problemas en la lectura ya que los textos muy delgados se pierden con el filtrado o se confunden con el fondo de color similar. Por ello es preferible mantener un texto uniforme. En el siguiente ejemplo se presenta un mismo tipo de letra en diferentes tamaños. Y lo que se puede verificar, es que la letra que es de tamaño pequeño prácticamente desaparece y la letra que es de tamaño grande, incluso cerca del límite aceptable también pierde sus características. Esto se debe a que como ya se mencionó anteriormente luego del filtrado puede haber pixeles que cambien sus características y quiten atributos a las letras.

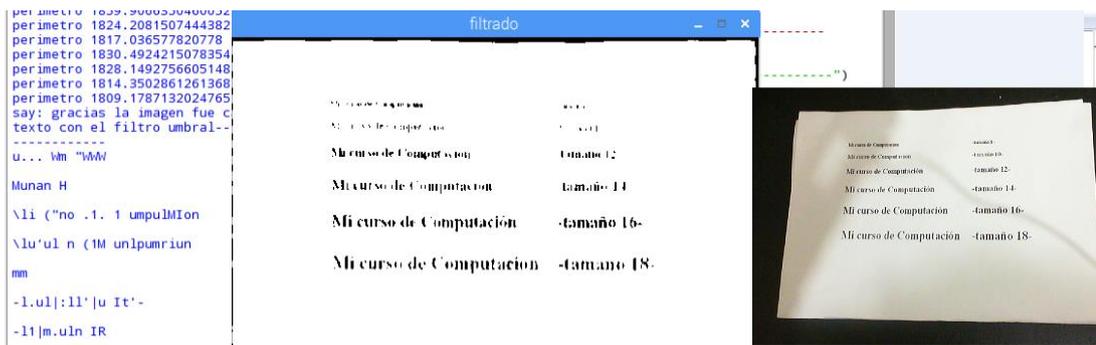


Figura 4.6. Idle de Python 3.4, letras delgadas pierden sus atributos.

Fuente: Autor.

Se continúa con las pruebas de campo, en donde al cumplir con las especificaciones recomendadas se verifica una reducción considerable del error en la lectura. En la siguiente figura se indica una prueba en donde el porcentaje de error es despreciable al cumplir con las especificaciones.

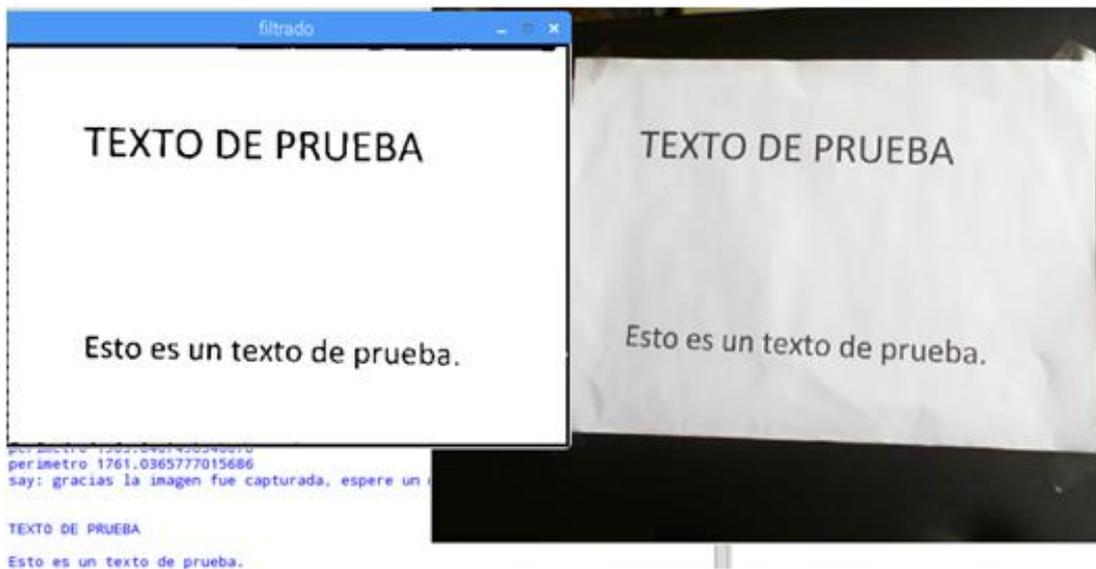


Figura 4.7. Idle de Python, resultados finales con un error despreciable.

Fuente: Autor.

De esta manera se cumple con los objetivos planteados, ya que se tiene un ayudante de lectura audible de texto impreso.

4.4.2. Desarrollo del Hardware

Se ve necesario la implementación de un sistema de control independiente del teclado y pantalla del sistema embebido para ampliar la utilidad del sistema. Por ello, se implementa un controlador compuesto por 4 pulsantes que será la interfaz entre el usuario y el sistema embebido. Por ello se establece la configuración de los pulsantes con resistencias en estado *Pull Down*, y con la ayuda de la herramienta de simulación Multisim se genera la siguiente simulación, conociendo que los pines GPIO del sistema embebido raspberry pi son tolerantes a 3.3V.

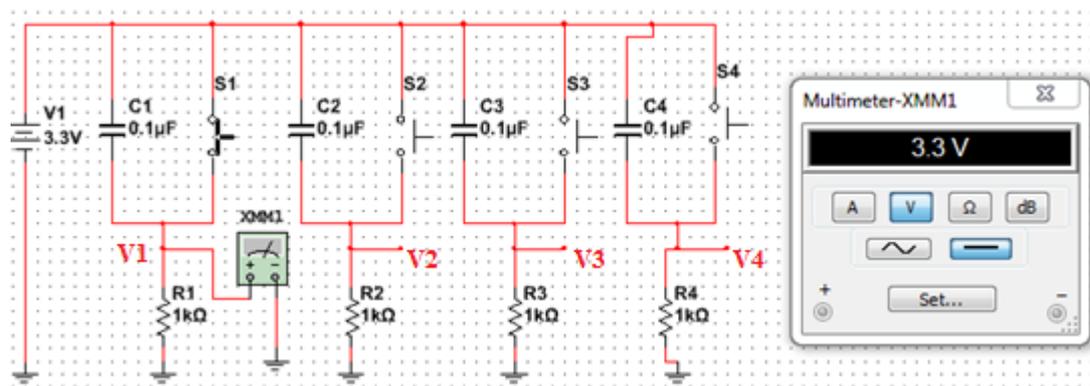


Figura 4.8. Software Multisim, simulación del circuito de control.

Fuente: Autor.

Se utiliza el esquema de la figura anterior en donde los condensadores cerámicos $C1=C2=C3=C4=0.1 \mu\text{F}$, son colocados en conexión paralelo de cada pulsante para minimizar el efecto rebote mediante hardware al pulsar. Por otra parte, se diseña con $R1=R2=R3=R4=1\text{k}\Omega$. Entonces se pretende no sobrepasar los límites de tensión y corriente de ingreso a la raspberry por lo que a alimentar con la fuente interna de la raspberry de 3.3V se garantiza que no sobrepase el nivel de tensión. Para la corriente, se realiza un cálculo mediante la ley de ohm y se obtiene.

$$I = \frac{V}{R} = \frac{3.3\text{V}}{1\text{k}\Omega} = 3.3\text{mA} \quad (2)$$

Entonces al tener:

$$V1=V2=V3=V4=3.3\text{V} \quad (3)$$

También se tiene:

$$I1=I2=I3=I4=3.3\text{mA} \quad (4)$$

Lo que garantiza que está dentro de los parámetros permitidos del sistema embebido Raspberry pi 3 analizados en el capítulo 1. Entonces, se procede a realizar el diseño de la placa de circuito impreso o conocido por sus siglas en inglés “Printed Circuit Board” (PCB), en la herramienta Ultiboard. Por consiguiente, se tienen el siguiente diseño.

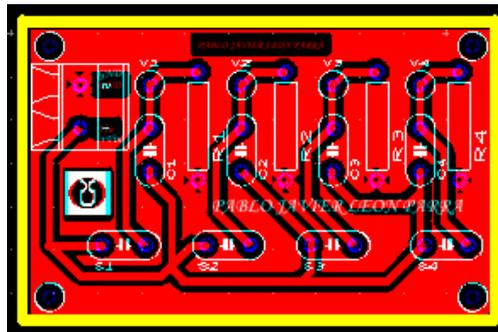


Figura 4.9. Software Ultiboard, diseño de PCB.

Fuente: Autor.

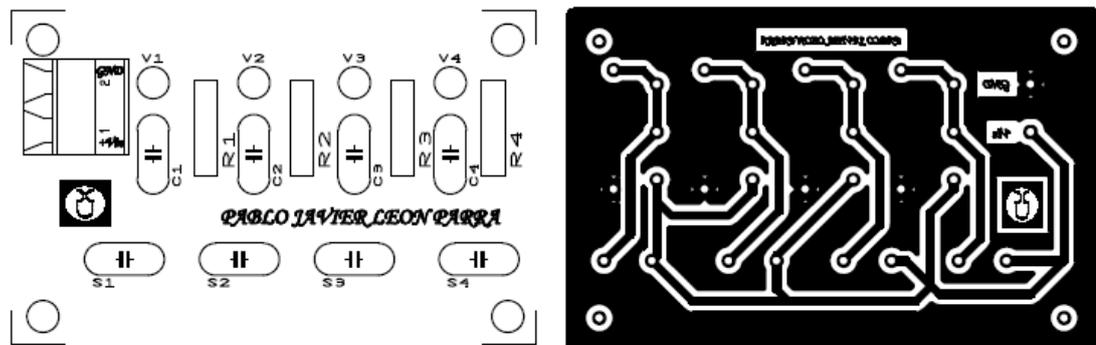


Figura 4.10. Software Ultiboard, diseño impreso del PCB.

Fuente: Autor.

En consecuencia, se obtiene el diseño terminado, impreso y listo para su posterior armado. Para simplificar el espacio se implementaron únicamente los 4 pulsante, los

mismos que se van a encargar de realizar todo el control y se le asignarán diferentes funciones a cada uno posteriormente.

Entonces, en el software se asignan a los pines GPIO los correspondientes pulsantes y su código establecido queda de la siguiente manera.

```
#####
#NUMERO DE GPIO RASPBERRY
A=24
B=23
C=17
D=27
#####
#Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(A,GPIO.IN)
GPIO.setup(B,GPIO.IN)
GPIO.setup(C,GPIO.IN)
GPIO.setup(D,GPIO.IN)
```

Figura 4.11. Idle de Python 3.4, asignación de pines GPIO.

Fuente: Autor.

Se asignaron los números de acuerdo con la numeración *GPIO.BCM* como se indica en la ilustración anterior, de tal manera que su ubicación sea cercana entre pines. La placa de control es conectada directamente al sistema embebido mediante conectores. En la siguiente figura se indica la placa de control implementada con su respectiva conexión al sistema embebido.

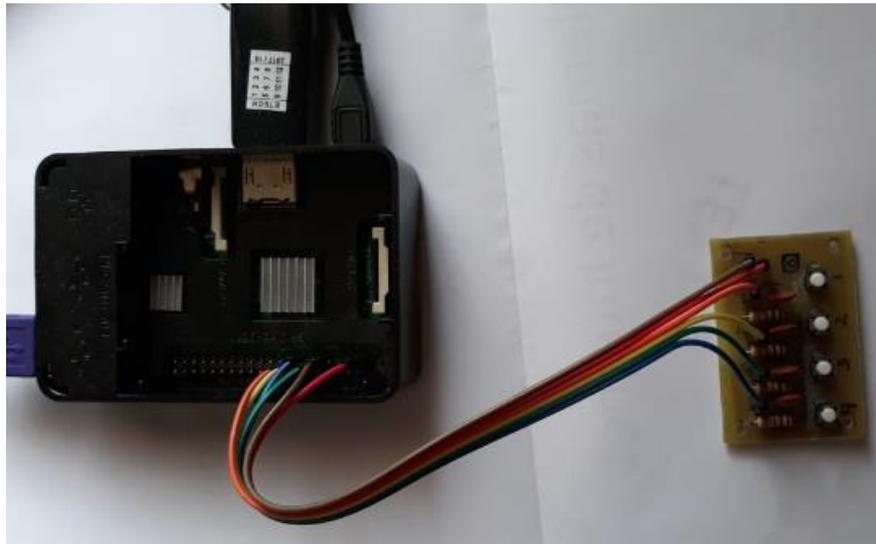


Figura 4.12. Conexión de la placa del control al sistema embebido.

Fuente: Autor.

4.4.3. Funcionamiento del sistema

También es necesario la implementación de un menú para que el usuario pueda interactuar y seleccionar la opción que desee realizar. En este caso se realiza un menú de 4 opciones que se indican a continuación.

```
print(" BIENVENIDOS AL AYUDANTE DE LECTURA \n")
vmenu(" BIENVENIDOS AL AYUDANTE DE LECTURA \n")
vmenu(" Presione: \n")
print("1 Para leer un nuevo texto.")
vmenu(" 1 Para leer un nuevo texto. \n")
print("2 Para escuchar el Último texto guardado.")
vmenu(" 2 Para escuchar el Último texto guardado. \n")
print("3 Para borrar todos los textos existentes.")
vmenu(" 3 Para borrar todos los textos existentes. \n")
print("4 Para salir.")
vmenu(" 4 Para salir. \n")
time.sleep(5) #Tiempo de espera antes de repetir la instrucciones.
```

Figura 4.13. Idle de Python 3.4, implementación del menú.

Fuente: Autor.

En consecuencia, se necesitan crear funciones para cada opción y funciones que determinen que opción se seleccionó que en este caso se asignará a la variable *llave*.

Para el primer caso que consiste en leer un nuevo texto, el programa asigna a la variable *llave=1*, la cual le permite entrar al proceso de lectura que se ha desarrollado a lo largo del documento, en donde inicializa la cámara y direcciona mediante alertas audibles para se enfoque la hoja, luego de eso se captura la imagen, se procesa en el sistema OCR y se sintetiza el texto. En caso de ya no querer leer más texto se tiene la opción de reproducir todo lo leído y se regresa al menú principal.

Para la opción número 2, se crea una función que permita escuchar los audios grabados en la última lectura. En la siguiente figura se detalla el algoritmo utilizado.

```
def escuchar():
    global llave
    global cantidad
    llave=0
    if (os.path.exists(directorio+"/audios")):#solo si existe el archivo, los borra
        cantidad=len(glob.glob(directorio+"/audios/*.mp3"))
        reproducir()
    else:
        print(" No existen audios. ")
        vmenu("9")
```

Figura 4.14. Idle de Python 3.4, implementación de las funciones correspondientes al menú.

Fuente: Autor.

Como se puede observar en la función anterior, solo si no existe el directorio se anuncia que no existen audios grabados, caso contrario el programa realiza una lectura de cuantos audios existen y envía a reproducir ordenadamente en una función creada anteriormente. Para las opciones 3 y 4, se tiene funciones similares en donde en la selección 3 borra toda la carpeta en caso de existir y en la opción 4 libera la variable *salir=0*, para que no continúe el programa y se siga repitiendo el bucle.

Entonces también se crean las funciones para cada pulsante, las cuales son encargadas de seleccionar entre el menú que se implementará luego y también realizar funciones extras. Por ejemplo, para el primer pulsante denominado pulsante A, al presionarlo también modifica los parámetros del reproductor pygame y pausa o reanuda el audio. Además, luego de cada lectura se puede determinar si se quiere seguir leyendo otro texto o pasar a escucharlo. Cabe recalcar que en el código se establece una línea que permite disminuir el rebote de los pulsantes por software ya que se establece que mientras esté pulsado se active un descanso del programa para que no active varias veces consecutivamente. El código se indica a continuación.

```
def PA(channel):
    global llave
    global paso
    if ((llave!=0)and(paso!=-1)):
        global tA
        global tD
        tD=0
        if GPIO.input(A):
            while GPIO.input(A):
                time.sleep(0.2)
                tA--tA
            if tA!=0:
                p.pause() # pausa
                print(" PAUSE ")
            else:
                p.unpause() # elimina la pausa
                print(" PLAY ")
        else:
            paso=0
            llave=1
```

Figura 4.15. Idle de Python 3.4, algoritmo para pulsantes de control.

Fuente: Autor.

Por consiguiente, los algoritmos de los demás pulsantes son semejantes a los de la función denominada PA, con las variaciones que con el segundo pulsante se puede controlar el volumen aumentándolo, con el tercer pulsante se disminuye el volumen y con el cuarto pulsante se detiene el audio y se sale del menú en caso de requerirlo.

En el anexo 1 se realiza un diagrama de bloques que simplifica el funcionamiento del sistema.

4.5. Conclusiones y limitaciones

Se cumple con el objetivo planteado, ya que se realiza la total integración de los sistemas que permiten realizar pruebas de campo y medir los resultados del asistente de lectura. Además, se puede conocer un aproximado de las limitaciones del equipo. Es por ello por lo que se resumen las principales características que se recomiendan tener en cuenta para minimizar el error en la lectura.

- Inicialmente se realizan pruebas en el menú y todas las funciones implementadas, por lo que se concluye que no existe ningún inconveniente al poder interactuar sin ningún problema con el sistema.
- Para la captura de las imágenes, en primera instancia es recomendable tener un texto de color negro sobre una hoja blanca con iluminación constante, tratando de evitar porciones de la hoja con sombra. Luego, para que se pueda detectar los contornos de la hoja con mayor facilidad y tomar la captura automática, se necesita disponer de un fondo de color oscuro, de preferencia de color negro mate para de esta manera resalte la hoja blanca y no se confundan los colores ni existan brillos que confundan el contorno.
- En cuanto a la resolución de la cámara, en este caso como ya se analizó anteriormente se utiliza una cámara Microsoft HD de 720p (720 líneas horizontales de resolución de pantalla de barrido progresivo) y 30 fonemas por segundo. Además, en base a pruebas de campo, se verifica que a una distancia aproximada entre los 40 cm y los 60 cm desde la hoja de tamaño A5 a la cámara existe un enfoque ideal para interpretar el texto. Por ello, se establece una resolución de 500 dentro del software en la biblioteca OpenCV, para poder mantener el espacio totalmente cubierto de la zona deseada.
- Tratando lo referente al tipo de texto, en el sustento teórico se recomienda evitar tipografías en manuscrita ya que complica la interpretación del sistema OCR. Además, se recomienda utilizar fuentes de trazo, es decir sin símbolos. También, dentro de las características de la fuente, es recomendable utilizar fuentes de tamaño grande. Por ejemplo, en base a las pruebas de campo se

determinó errores despreciables casi nulos en fuentes constantes de 36 puntos o superiores. Por consiguiente, según el sustento teórico planteado en el capítulo 1, la medida referencia que se le da las fuentes es un promedio de 0.035 cm por punto con variaciones entre tipos de letras. Pero, de ser el caso con 36 puntos daría un tamaño de 1.26 cm y la letra que se utilizó para las pruebas es *Times New Roman* que tiene una altura de 0.8 cm en tamaño 36 puntos. Por lo tanto, se recomienda utilizar letras que cumplan estas características o las superen en 0.8 cm de alto.

- En lo relacionado con la capacidad del sistema embebido se concluye que soporta un límite superior al necesario. Esto se debe a que, con lo referente a los límites de palabras almacenadas en una variable, está limitado únicamente por la cantidad de memoria disponible en el dispositivo donde se ejecuta el programa. En este caso se dispone de 32 GB de memoria externa y suponiendo que exista un espacio libre de 20 GB para almacenamiento al requerir solo 1 byte por cada carácter, la capacidad se vería limitada únicamente por la cantidad de caracteres que se incluyan en una hoja y en el número de hojas. Ahora bien, mediante pruebas se constató que dentro de una hoja de tamaño A5 se pueden incluir un máximo de 139 caracteres con el tipo de letra *Times New Roman* 36 puntos que entran en 6 líneas. En consecuencia, teóricamente se podría almacenar más de 143 millones de hojas con estas características. Por lo tanto, la capacidad de almacenamiento de caracteres en este caso no es una limitante para este sistema.
- Además, si se trata de la capacidad de almacenamiento en archivos de audio, se sabe mediante pruebas de campo que los audios generados con las características del sistema ocupan un espacio aproximado de 8 Kb por cada 20 palabras generadas. En consecuencia, si se trataría de un audio que contenga el máximo de palabras calculado se estaría tratando de alrededor de 52 Kb por hoja. Tomando en cuenta estas aproximaciones y con la misma capacidad de 20 GB libres de almacenamiento se podría almacenar teóricamente hasta más

de 384 mil audios de páginas completas. Es decir, tampoco se considera un limitante en este caso.

- Finalmente, se analiza el consumo energético del sistema implementado, ya que se tiene como dato según el análisis del capítulo 1, un consumo aproximado en pleno rendimiento del sistema embebido Raspberry pi 3 de 350mA. Entonces al alimentar con un sistema de baterías de capacidad 6000 mAh, se podría tener teóricamente cerca de 17 horas alimentación. Entonces, se podría considerar como un limitante en caso de un uso prolongado fuera de un sistema de alimentación fijo o carga de baterías.

CONCLUSIONES

- Se obtiene un resultado muy satisfactorio ya que se han cumplido todos los objetivos planteados al desarrollar un ayudante de lectura mediante visión artificial.
- Se concluye que el sistema OCR tiene mayor facilidad el reconocimiento de caracteres cuando se trata de palabras enteras y no solo letras, ya que la información del contexto aporta a relacionar las letras adyacentes al contenido que se está analizando.
- Los módulos texto a voz o conocidos como TTS, tienen cierto grado de error en relación con el texto que se va a interpretar y requieren una constante actualización para minimizar ese error.
- Se verifica que el uso del lenguaje de programación Python fue completamente satisfactorio, gracias a su fácil implementación, compatibilidad y gran alcance, no se encontró ningún inconveniente en su utilización. También, la potente biblioteca OpenCV es compatible con el lenguaje Python y dispone de todas las características necesarias para adaptar visión artificial al ayudante de lectura deseado.
- En la captura de la imagen, ha dado mejores resultados cuando se trata de hojas de tamaño A5, ya que se puede acercar más la cámara y permite visualizar mejor el texto. Además, es ideal tomar la fotografía en condiciones de luz constantes, ya que, al tener 2 diferentes tonos de luz o una porción de sombra sobre la imagen, puede generar error en el procesamiento de la imagen y esto maximiza el error en la lectura. En cuanto al tipo de letra se sugiera que sean tipos de letras sin símbolo de tamaños iguales o superiores a los 0.8 cm de alto y evitar manuscritos, como por ejemplo *Times New Roman 36 puntos*. La captura generó mejores resultados cuando fue tomada a distancias entre 40 cm a 60 cm entre la hoja blanca con letras negras y la cámara, siendo que la hoja se encontró en un fondo negro mate.

- Es de vital importancia cuando se desconecte la pantalla del sistema embebido, no visualizar con el comando *imshow* de OpenCV dentro del programa, ya que las pruebas finales fueron tomadas solo con la ayuda auditiva y el comando *imshow* al no disponer de una pantalla generaba un error que no le permitía avanzar. Posteriormente, solo al quitar esta instrucción se corrigió el problema.
- Es recomendable liberar los recursos de OpenCV cuando se termine la lectura de la imagen. Por ello, al salir utilizar los comandos *cap.release()* y *cv2.destroyAllWindows()*. De lo contrario, al quedarse utilizando recursos innecesarios, puede hacer lento el procesamiento del programa.
- Se concluye que el dispositivo utilizado Raspberry pi 3 con 64 Bytes, 1 Gb de RAM y una memoria externa de 32 GB soporta un límite superior al necesario, ya que, los requerimientos para almacenar la información leída se ven limitado únicamente por la cantidad de memoria disponible en el dispositivo y en este caso sobrepasa al necesario.

RECOMENDACIONES

- No se recomienda el reconocimiento de caracteres con textos impresos a mano, ya requieren algoritmos extras, por lo que dificulta su interpretación, es por

ello por lo que se tratará solo el reconocimiento de textos impresos en determinada letra computarizada.

- Se verificó la gran utilidad que presenta el software OCR Tesseract y sus amplias y potentes aplicaciones, ya que es diseñado y en desarrollo por Google. Además, con código abierto para plataformas como Windows, Mac OS X, Linux y OS/2; por lo que se recomienda la utilización de este software.
- Se recomienda la utilización del sistema embebido Raspberry Pi, ya que dispone de las características necesarias para la implementación del ayudante de lectura y su gran compatibilidad con sistemas operativos, su accesibilidad y su costo accesible lo hacen aptos para esta aplicación.
- También es recomendable el uso del lenguaje de programación Python, ya que es libre y con las capacidades necesarias para esta implementación. Además, al tener una amplia compatibilidad, si es compatible con la potente librería de visión artificial OpenCV necesaria y acorde a la necesidad de este sistema a implementar.
- Para el sistema OCR, se recomienda que la imagen a procesar sea clara, concreta y amplia en lo posible, ya que de esta manera el error de reconocimiento es mucho menor. Por lo tanto, en la segmentación es indispensable enfocarse al texto a leer y extraer esa porción de imagen.
- Es completamente indispensable, que la hoja que contenga el texto que se va a leer se coloque sobre un fondo de un color opuesto al de la hoja, ya que de esta manera se ayuda a la detección de contornos. Se recomienda en caso de tener una hoja blanca con letras negras, que se coloque sobre un fondo de color oscuro, de preferencia negro mate, ya que el color brillante puede confundir el contorno. Además, una recomendación muy importante es mantener el texto a un nivel de luz constante, ya que la sombra parcial en un texto genera distorsión en el momento del procesado de la imagen y puede causar error en el OCR.

- Es recomendable también, mantener la calidad de las imágenes, por ello se sugiere guardar las capturas en formato *tif*, en vista de que aporta a mantener una calidad adecuada en la imagen.
- Por otra parte, se recomienda utilizar generadores de voz independientes de internet en aplicaciones portátiles, ya que, genera lentitud y dependencia de conexión a internet, por lo que, a pesar de disponer de una voz diferente, un tanto más robótica y con un todo más sintético se utiliza un módulo libre.

REFERENCIAS

Abedin, I. (2018). *journaldev*. Obtenido de <https://www.journaldev.com/16140/python-system-command-os-subprocess-call>

- Aldabas-Rubira, E. (2002). Introducción al reconocimiento de patrones mediante redes neuronales. *IX Jornades de Conferències d'Enginyeria Electrònica del Campus de Terrassa*. Terrassa, España.
- Alvear Puertas, V., Rosero Montalvo, P., Peluffo Ordóñez, D., & Pijal Rojas, J. (2017). Internet de las Cosas y Visión Artificial, Funcionamiento y Aplicaciones: Revisión de Literatura. *Enfoque UTE*, 8(1), 244-256.
- aprenderpython. (2018). *aprenderpython.net*. Obtenido de <https://www.aprenderpython.net/comenzando-con-contornos/>
- Athanasias, D. (14 de mayo de 2014). *wiki.python.org*. Obtenido de <https://wiki.python.org/moin/NumPy?highlight=%28numpy%29>
- Belmonte , J. G. (2014). *Universidad Nacional de Rosario*. Obtenido de Unidad Técnica de Informática del Departamento de Sistemas e Informática (DSI): https://www.dsi.fceia.unr.edu.ar/images/Sistemas_Embebidos/SEA_Unidad1.pdf
- Bembibre, C. (11 de mayo de 2011). *Importancia.org*. Obtenido de <https://www.importancia.org/lectura.php>
- Beyeler, M. (2017). *Machine Learning for OpenCV*. Birmingham, Mumbai: Packt.
- Biblioteca de la Universidad de Extremadura. (27 de Septiembre de 2018). *Servicio de bibliotecas*. Obtenido de <https://biblioguias.unex.es/c.php?g=572102&p=3944889>
- Burger, D. (1994). Improved Access to Computers for the Visually Handicapped: New Prospects and Principles. *IEEE Transactions on Rehabilitation Engineering*, 2(3), 111-118.
- Carrio Díaz, M. (17 de enero de 2006). *Ministerio de Educación, Cultura y Deporte. (Gobierno de España)*. Obtenido de <http://recursostic.educacion.es/observatorio/web/fr/equipamiento-tecnologico>

- Cávez, A. B. (mayo de 1997). *Universitat Politècnica de Catalunya*. Obtenido de <https://upcommons.upc.edu/bitstream/handle/2099/9620/Article016.pdf?sequence=1&isAllowed=y>
- Challenger-Pérez, I., Díaz-Ricardo, Y., & Becerra-García, R. A. (2014). El lenguaje de programación Python. *Ciencias Holguín*, 20(2), 1-13.
- CONADIS, C. N. (2018, agosto). *Personas con discapacidad registradas*. Obtenido de <https://www.consejodiscapacidades.gob.ec/wp-content/uploads/downloads/2018/08/persona.html>
- Cruz, J. M. (2013). Sistemas embebidos, sistemas de tiempo real. *Simposio argentino de sistemas embebidos* (págs. 5 - 8). Fiuba, Buenos Aires: s.e.
- Del Valle Hernandez, L. (2018). *programarfacil.com*. Obtenido de <https://programarfacil.com/podcast/81-vision-artificial-opencv-phyton/>
- Departamento de Tecnología IES La Cabrera. (2012). *IES La Cabrera*. Obtenido de <http://cuartoinformatica.tecnosjulio.com/wp-content/uploads/2013/01/La-imagen-digital.pdf>
- Diccionario de la lengua española. (2017). *Real Academia Española*. Obtenido de <http://dle.rae.es/?id=N3m3mKb>
- Fernández del Campo, J. E. (2001). *DESAFÍOS DIDÁCTICOS DE LA LECTURA BRAILLE*. Madrid: Carácter, S.A.
- Fundación de Software Python. (2018). *Fundación en Python.org*. Obtenido de <https://www.python.org/>
- Fundación Raspberry Pi. (2018). *Raspberry Pi*. Obtenido de <https://www.raspberrypi.org/?s=raspberry+pi+3>
- Garzón Canchignia, R. C., & Pacheco Gavilánez, J. A. (2016). *Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE*. Obtenido de Departamento de Eléctrica y Electrónica :

<https://repositorio.espe.edu.ec/bitstream/21000/11978/1/T-ESPEL-ENI-0379.pdf>

Geekland. (7 de jun de 2015). *Geekland*. Obtenido de <https://geekland.eu/fundamentos-usos-limitaciones-oct/>

Giusti, M. R., Lira, A. J., Rodríguez Vuan, J. P., & Villareal, G. L. (2016). Accesibilidad de los contenidos en un repositorio institucional: análisis, herramientas y usos del formato EPUB. *e-Ciencias de la Información*, 6(2), 1-23.

González, A. F., & Florez López, J. C. (s.f.). *Ministerio de Educación, Cultura y Deporte. (Gobierno de España)*. Obtenido de Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado: http://www.ite.educacion.es/formacion/materiales/6/cd/m3/tamao_de_las_fuentes.html

Grupo MINA. (2017). *Universidad de la República de Uruguay*. Obtenido de Instituto de Computación de la Facultad de Ingeniería: https://eva.fing.edu.uy/pluginfile.php/88550/mod_resource/content/1/sistemas%20embebidos.pdf

Guido van Rossum. (octubre de 2017). *Tutorial de Python*. (J. Fred L. Drake, Ed.) Argentina: Copyright, Python Software Foundation. Obtenido de <http://docs.python.org.ar/tutorial/>

Gutierrez, R., Frydson, M. F., & Phd. Vintimilla, B. (2011). *Aplicación de Visión por Computador para el Reconocimiento Automático de*. Guayaquil, Guayas, Ecuador.

Instituto de Astrofísica de Andalucía. (s.f.). *Instituto de Astrofísica de Andalucía*. Obtenido de https://www.iaa.csic.es/python/Iniciacion_Python-Dia1.pdf

- Izzad, R., Nursuriati, J., Noraini, S., & Norizah, A. (2015). An Improved Syllabification for a Better Malay Language Text-to-Speech Synthesis (TTS). *Procedia Computer Science*, 76, 417-424.
- lame.sourceforge. (2018). *lame.sourceforge*. Obtenido de <http://lame.sourceforge.net/index.php>
- Llisterri, J. (15 de febrero de 2018). *Universidad Autónoma de Barcelona*. Obtenido de Departamento de Filosofía Española: http://liceu.uab.es/~joaquim/speech_technology/tecnol_parla/synthesis/format_synthesis/sintesis_formantes.html
- Martinez, A. (22 de septiembre de 2016). *La tecnología*. Obtenido de <http://aprendiendoconntecnologia.blogspot.com/>
- Martos Navarro, F., & Muñoz Labiano, Á. M. (2008). *Agentes de movilidad del ayuntamiento de madrid. Test*. Madrid: MAD.
- Navacerrada, J. (25 de octubre de 2017). *Universidad Politécnica de Madrid*. Obtenido de http://oa.upm.es/51869/1/TFG_JORGE_NAVACERRADA.pdf
- Oleagordia Aguirre, I. J. (mayo de 2012). *Universidad del país vasco*. Obtenido de https://ocw.ehu.eus/pluginfile.php/2869/mod_resource/content/1/GENERAL/Empleo_del_PC_en_la_IP.pdf
- opencv dev team. (10 de noviembre de 2014). *docs.opencv.org*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html
- OrCam, T. (julio de 2018). *Orcam*. Obtenido de <https://www.orcam.com/es/>
- Ordóñez L., J. P. (2009). Reconocimiento óptico de caracteres (OCR) con redes neuronales. Loja, Loja, Ecuador.
- Pérez González, M. S., Lara Lemus, Y., & Naranjo Hernández, D. (2004). *La digitalización de textos, base de la biblioteca digital: su impacto como apoyo*

a la docencia en la Universidad de las Ciencias Informáticas. La Habana: Editorial Universitaria.

Pérez Porto, J., & Gardey, A. (2012). *Definicion.de*. Obtenido de <https://definicion.de/lectura/>

Programiz. (s.f.). *Programiz*. Obtenido de <https://www.programiz.com/python-programming/string>

Python Software Foundation. (2018). *Python Package Index*. Obtenido de <https://pypi.org/project>

Quoc Vuong, B., & Ngoc Do, H. (2014). Diseño e implementación del lector de tarjetas de nombre en varios idiomas en la plataforma Android. *Conferencia internacional sobre tecnologías avanzadas para las comunicaciones (ATC 2014)* (pág. 1). Hanoi, Vietnam: IEEE. doi:10.1109 / ATC.2014.7043445

Rodríguez Fuentes, A. (2005). *¿Cómo leen los niños con ceguera y baja visión?* Málaga: Aljibe.

Rodríguez, A., & Gallego Granda, J. L. (2001). Potencial educativo de las nuevas tecnologías en la lectoescritura de personas con deficiencia visual. *Comunicar: revista científica iberoamericana de comunicación y educación*, 9(17), 158-164.

Roman Bueno, J. C., & Gonzalez Mantilla, K. J. (2016). *Universidad Industrial de Santander*. Obtenido de Escuela de Ingeniera de sistemas e Imformtica: <http://wiki.sc3.uis.edu.co/images/e/e1/GR7.pdf>

Rose, D. (20 de febrero de 2012). *BBC*. Obtenido de https://www.bbc.com/mundo/noticias/2012/02/120216_braille_ciegos_uso_de_clive_cch

Rubio, V. T. (junio de 2015). *Universidad Politécnica de Cartagena*. Obtenido de Escuela Técnica Superior de Ingeniería de Telecomunicación:

<http://repositorio.upct.es/bitstream/handle/10317/4835/pfc6249.pdf;sequence=1>

s.a. (27 de mayo de 2012). *wiki.python.org*. Obtenido de <https://wiki.python.org/moin/>

Salas Arriarán, S. (2015). *Todo sobre sistemas embebidos: Arquitectura, programación y diseño de aplicaciones prácticas con el PIC18F*. Lima-Perú: Universidad Peruana de Ciencias Aplicadas (UPC).

Sánchez, J. M. (19 de junio de 2013). *OrCam, unas gafas para ciegos capaces de leer lo que éstos le señalan. ABC-Tecnología*. Obtenido de <http://www.abc.es/tecnologia/informatica/20130619/abci-orcam-gafas-leen-solas-201306181544.html>

Serrano Mascaraque, E. (2008). Accesibilidad web para los discapacitados: ¿una nueva herramienta para la integración social o un nuevo motivo de exclusión social? *Ibersid: revista de sistemas de información y documentación.*, 2(1), 23-31.

Singh, M. (2017). *Geeksforgeeks*. Obtenido de <https://www.geeksforgeeks.org/byte-objects-vs-string-python/>

Slashdot Media . (2018). *SourceForge*. Obtenido de <http://espeak.sourceforge.net/>

Smith Torres, R. (abril de 2009). *Conversión de Texto a Voz Mediante Reglas y Redes Neuronales: Traducción de Texto a fonemas más acentuación y puntuación*. Santiago de Chile, Chile.

Tarín, P. (22 de noviembre de 2017). *infotecarios*. Obtenido de <http://www.infotecarios.com/aplicaciones-conversion-texto-voz/#.W7vsDHtKjIU>

Tecnología informática. (s.f.). *Tecnología informática*. Obtenido de <https://tecnologia-informatica.com/tipos-licencias-software-libre-comercial/>

Velasco, R. (2 de marzo de 2017). *redeszone*. Obtenido de <https://www.redeszone.net/2017/03/02/energia-consumida-raspberry-pi/>

Vélez Serrano, J. F., Moreno Díaz, A. B., Sánchez Calle, A., & Sánchez Marín, J. L. (2003). *Visión por computador*. Madrid: Dykinson.

Vialfa, C. (27 de julio de 2017). *es.ccm.net*. Obtenido de <https://es.ccm.net/contents/724-el-formato-tif>

Walker, J. S. (2018). *Python: La Guía Definitiva para Principiantes para Dominar Python*. (A. Martinez, Trad.) s.l.: Babelcube Inc.

Zaganelli, G. (21 de Abril de 2014). *Diccionario Digital de Nuevas Formas de Lectura y Escritura*. Obtenido de <http://dinle.usal.es/searchword.php?valor=Lectura%20visual>

Zorrilla, G. A. (2014). *pygame.org*. Obtenido de <https://www.pygame.org/docs/index.html>

ANEXOS

Anexo 1: Diagrama de flujo

